SEQUENT

DYNIX/ptx User's Guide

1003-48605-01

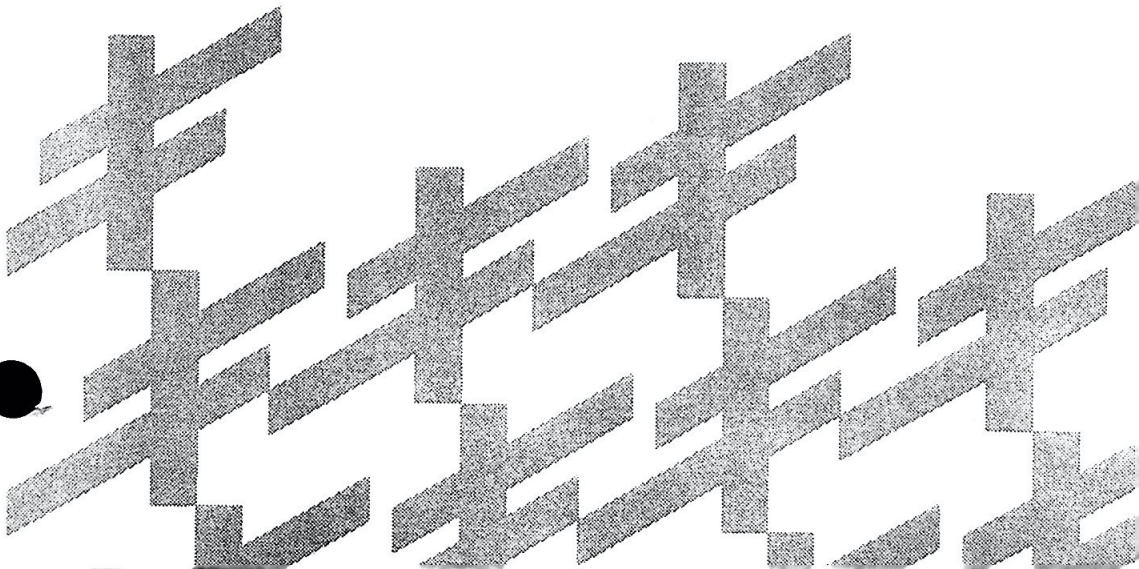
SEQUENT

THIS SHEET LIFTER
MUST BE PLACED ON
TOP OF CONTENTS

GRAPHIC PRODUCTS
Chicago, Illinois 60609

DYNIX/ptx®
User's Guide

1003-48605-01
Sequent Computer Systems, Inc.



Sequent, DYNIX and DYNIX/ptx are registered trademarks of Sequent Computer Systems, Inc.

ptx/NFS is a trademark of Sequent Computer Systems, Inc.

UNIX is a registered trademark of AT&T.

NFS is a trademark of Sun Microsystems, Inc.

Copyright © 1989, 1990 by Sequent Computer Systems, Inc. All rights reserved. This document may not be copied or reproduced in any form without permission from Sequent Computer Systems, Inc. Information in this document is subject to change without notice.

Printed in the United States of America.

Contents

About This Guide

Chapter 1. The Operating System

1.1 Introduction 1-1
1.2 The Kernel 1-1
1.3 The Shell 1-2
1.4 Filesystems 1-3
1.5 Commands 1-5

Chapter 2. Getting Started

2.1 Introduction 2-1
2.2 The Terminal 2-1
2.3 Logging into the System 2-4
2.4 Entering Commands 2-10
2.5 Logging Off 2-11

Chapter 3. Using the Filesystem

3.1 Introduction 3-1
3.2 Managing Directories and Files 3-11
3.3 Changing to Another Group: newgrp 3-58
3.4 Disk Quotas 3-59

Chapter 4. Using the Shell

4.1 Introduction 4-1
4.2 Filename Substitution 4-2
4.3 Running Commands in the Background 4-6
4.4 Executing Commands Sequentially 4-7
4.5 Input and Output Redirection 4-8
4.6 Printing Arguments: echo 4-13
4.7 Using Command Output as an Argument 4-14
4.8 Removing Meanings from Special Characters 4-14
4.9 The Bourne and Korn Shell Environment 4-20
4.10 Using the Korn Shell 4-28
4.11 Using the C Shell 4-37

Chapter 5. Online Resources

5.1 Introduction 5-1
5.2 Online Man Pages 5-1
5.3 The help Command 5-3
5.4 ASSIST 5-6

Chapter 6. Using the Printer
6.1 Introduction 6-1
6.2 The LP Print Service 6-1
6.3 Printing a File: lp 6-2
6.4 Printing Command Output 6-9
6.5 Displaying LP Status: lpstat 6-10
6.6 Changing a Print Request 6-13
6.7 Canceling a Print Request: cancel 6-14
6.8 Enabling and Disabling Printers: enable, disable 6-15

Chapter 7. Managing Processes
7.1 Introduction 7-1
7.2 Creating Processes 7-1
7.3 Managing Processes in the Bourne Shell 7-2
7.4 Managing Processes in the Korn Shell or C Shell 7-4
7.5 Running Commands at a Later Time: batch, at 7-7
7.6 Running Processes after Logout: nohup 7-10

Chapter 8. The ed Line Editor
8.1 Introduction 8-1
8.2 Invoking ed 8-8
8.3 Writing to a File and Quitting ed 8-9
8.4 Displaying Text 8-12
8.5 Locating Lines in a File 8-14
8.6 Displaying Help Messages 8-17
8.7 Creating Text 8-18
8.8 Undoing the Previous Command 8-21
8.9 Deleting Lines 8-21
8.10 Searching for Text 8-22
8.11 Substituting Text 8-25
8.12 Matching Patterns 8-29
8.13 Moving Text 8-35
8.14 Miscellaneous Features 8-38

Chapter 9. The vi Screen Editor
9.1 Introduction 9-1
9.2 Invoking vi 9-9
9.3 Writing to a File and Quitting vi 9-11
9.4 Creating Text 9-14
9.5 Moving the Cursor 9-17
9.6 Undoing Commands 9-30
9.7 Deleting Text 9-30
9.8 Modifying Text 9-34
9.9 Cutting and Pasting Text 9-41

9.10 Miscellaneous Commands 9-44
9.11 Modifying Your vi Environment 9-48

Chapter 10. Communications
10.1 Introduction 10-1
10.2 Reading System-Wide News 10-1
10.3 Sending Messages to Another User's Terminal 10-2
10.4 Using mail 10-4
10.5 Using mailx 10-10
10.6 Using UUCP 10-41

Appendix A. ASSIST Error Messages

Glossary

Index

Figures

1-1 The operating system 1-2
1-2 The root filesystem 1-4
1-3 Executing a command 1-8

3-1 Home directories in the filesystem 3-2
3-2 The *starship* directory 3-5
3-3 Output from the **ls –l** command 3-19

10-1 Sample *.mailrc* file 10-34

About This Guide

Overview

This manual introduces the DYNIX/ptx operating system. It describes the components of the operating system, tells how to start up and quit an operating system session, and describes several operating system commands. It also describes the shell programs available on the system and tells how to tailor them to meet your specific needs.

This guide also tells how to use these operating system programs:

- The print service
- Text editors
- Electronic mail programs
- Programs providing online assistance

Assumptions About the Reader

This guide is intended for system administrators, programmers, and other users working with the DYNIX/ptx operating system. It assumes that the reader has some familiarity with computers, but is new to the operating system or to some of its features.

Notation

The following notational conventions are used in this manual:

- Filenames and user-defined parameters in text are shown in *italics*.
- System output is shown in `even-width` font.
- Commands in text and examples of input to be entered literally are shown in **bold**.
- Environment variables are shown in **bold**.

Chapter 1
The Operating System

1.1 Introduction 1-1

1.2 The Kernel 1-1

1.3 The Shell 1-2

1.4 Filesystems 1-3
Files 1-3
The Filesystem Structure 1-4

1.5 Commands 1-5
The Shell Command Line 1-6
Executing a Command 1-7

Figures

1-1 The operating system 1-2
1-2 The root filesystem 1-4
1-3 Executing a command 1-8

Chapter 1
The Operating System

1.1 Introduction

The operating system controls the computer. It allocates system resources, acts as the link between you and the computer, and provides tools to help you do your work. It offers the following features:

- A general-purpose environment that allows you to perform a wide variety of jobs or applications
- An interactive environment that allows you to communicate directly with the computer and to receive immediate responses to your requests
- A multi-user environment that allows many users to share the computer's resources
- A multi-tasking environment that can execute several programs simultaneously

This chapter describes three major components of the operating system: the kernel, the shell, and the filesystem. It also tells how to enter commands.

1.2 The Kernel

The kernel is the nucleus of the operating system. It controls access to the computer, handles devices such as disk and tape drives, manages the computer's memory, maintains the filesystem, and allocates the computer's resources among users. The kernel works invisibly; you don't need to be aware of it while doing your work.

1.3 The Shell

The shell is a command interpreter that enables you to communicate with the operating system. When the shell is waiting for you to enter a command, it will display a prompt (typically a dollar or percent sign) on your screen. The shell reads the commands you type and interprets them as requests to execute programs, to access files, or to provide output. Because the shell reads input from you and sends messages to you, it is described as interactive.

The shell is also a programming language that provides conditional execution and control flow features.

Figure 1-1 shows a model of the operating system.

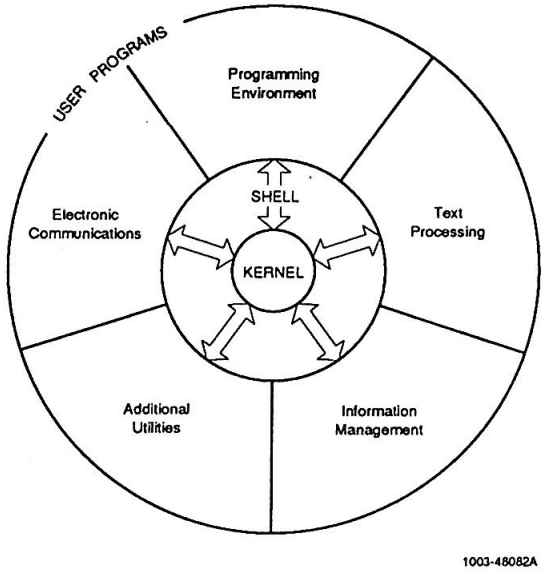


Figure 1-1. The operating system. The circles represent the kernel, the shell, and user programs or commands. The arrows represent the shell's role as an interpreter between you and the kernel.

There are three shell programs available to you: the Bourne shell, the Korn shell, and the C shell. Basically, these programs all execute your commands in the same way. However, the Korn and C shells provide some additional features such as the ability to reexecute commands. Chapter 4 describes the differences among these shell programs.

1.4 Filesystems

Filesystems provide a logical method to organize, retrieve, and manage information. Each filesystem consists of a set of files arranged in a hierarchical structure.

1.4.1 Files

The file is the basic unit in a filesystem. There are three types of files: ordinary files, directories, and special files. However, to the operating system, all files are alike.

Ordinary Files An ordinary file is a collection of characters the operating system treats as a unit. Ordinary files can store any information you want to save. For example, they can contain text for letters or reports, or code for the programs you write. After you create a file, you can add material to it, delete material from it, or remove it entirely when it is no longer needed.

Directories A directory is a super-file containing a group of related files. For example, a directory called *sales* could hold files named *jan*, *feb*, and *mar* that contain monthly sales figures. You can create directories, add new files to them, delete existing files, or remove the directories themselves at any time.

The directories you create are located in your home directory. This directory was assigned to you when you were set up as an authorized user on the system. You control your home directory; you can set permissions that determine who can read, change, or execute your files. You also determine the structure of your home directory; you can create any other directories and files you want within your home directory.

Special Files A special file represents a physical device such as a terminal, disk drive, tape drive, or communication link. The system reads and writes to special files in the same way it reads and writes to ordinary files. However, the system's read and write requests do not activate the normal file access mechanism. Instead, they activate the device handler associated with the special file.

1.4.2 The Filesystem Structure

The structure of a filesystem is hierarchical; it represents an an inverted tree. Figure 1-2 shows a portion of the root filesystem, which is the starting point for the operating system itself and is located at the top of the hierarchy.

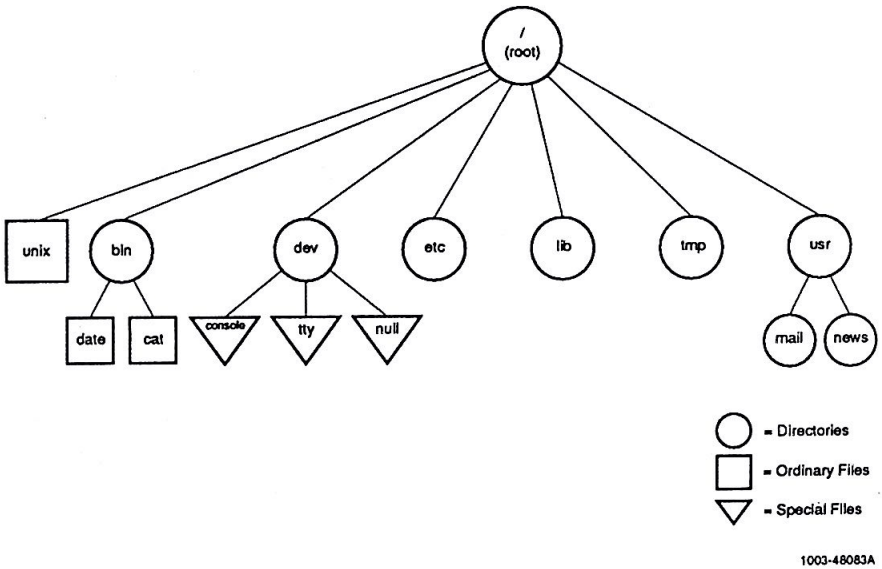


Figure 1-2. The root filesystem.

The root directory is the starting point for the root filesystem. It is designated by a slash (/), as shown in Figure 1-2. All directories and files are arranged hierarchically under the root directory, including *unix*, the file containing the kernel. The operating system also maintains several directories for its own use. These directories (called system directories) are located on the first level below the root directory. They store the following information:

/bin Contains executable programs and utilities

/dev	Contains special files representing peripheral devices such as the console, line printers, user terminals, and disks
/etc	Contains programs and data files for system administration
/lib	Contains libraries for programs and languages
/tmp	Contains temporary files that any user can create
/usr	Contains other directories such as *mail*, which stores files containing electronic mail, and *news*, which stores files containing news items

The operating system can also include other filesystems that are attached to the root filesystem. In most cases, you don't need to be aware of these filesystems. They simply appear as subdirectories within the root filesystem.

1.5 Commands

A *program* is a set of instructions for the computer. A *command* is the name of a program. When you want to run a program, you issue the command that corresponds to that program.

Packages of programs are called *tools*. As a typical user of the operating system, you have many standard programs and tools available to you. If you use the operating system to write programs and develop software, you can also draw on system calls, subroutines, and other tools.

The outer circle in Figure 1-1 organizes the operating system programs and tools into these categories:

Text processing
: The system provides programs such as **ed** (the line editor) and **vi** (the screen editor) for creating and changing text, a spelling checker for locating spelling errors, and text formatters for producing printed copies.

Information management
: The system provides many programs that allow you to create, organize, and remove files and directories and to obtain information from them.

Electronic communication
: The system contains programs such as **mail** that enable you to transmit information to other users.

Programming environment
: The system provides interfaces between the operating system and programming languages and supplies many utility programs.

Additional utilities
: The system offers capabilities for generating graphics and performing calculations.

1.5.1 The Shell Command Line

To make your requests understandable to the shell, you must enter each command in the correct format, or command line syntax. This syntax defines the order in which you type the components of a command.

Command lines have this syntax:

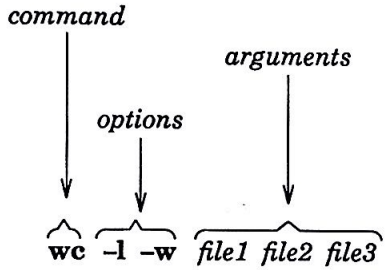
command option(s) argument(s)

- A *command* is the name of the program you want to run.
- An *option* modifies how the command runs.
- An *argument* specifies data on which the command is to operate (usually the name of a directory or file).

You must press Return at the end of a command line. The shell then executes the command.

When you include options or arguments in a command line, be sure to separate the options or arguments by at least one space. If an argument name contains a blank, enclose the argument in quotation marks. For example, if the argument to your command is **sample 1**, type it as "**sample 1**" or '**sample 1**'. If you omit the quotation marks, the shell will interpret **sample** and **1** as two separate arguments.

Some commands allow you to specify multiple options and arguments. For example, the **wc** command shown on the next page includes two options and three arguments.



You must precede most options with a hyphen (–). If you specify more than one option, the operating system usually allows you to group them together. In the previous example, you could type the options as **-lw**. The example also specifies three files (*file1*, *file2*, and *file3*) as arguments. You must leave a space between arguments.

The *Reference Manual* describes the correct syntax for each operating system command.

1.5.2 Executing a Command

To execute a command, type a command line at the shell prompt (`$` for the Bourne and Korn shells, `%` for the C shell) that appears on your screen. Then press Return. The shell considers your command as input, searches through one or more directories to retrieve the program you specified, and conveys your request and the specified program to the kernel. The kernel then follows the instructions in the program and executes the command. When the program finishes running, the shell will display another command prompt (`$` or `%`) to signal that it is ready for your next command.

Figure 1-3 shows the flow of control when the operating system executes a command.

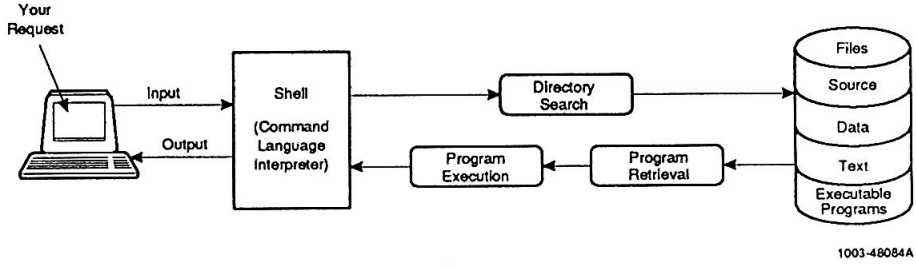


Figure 1-3. Executing a command.

Chapter 2
Getting Started

2.1 Introduction 2-1

2.2 The Terminal 2-1
Keyboard Characteristics 2-2
Typing Speed 2-2
Correcting Typing Errors 2-2
Deleting the Current Line 2-2
Deleting Characters 2-3
Using a Delete Character as Text 2-3
Control Characters 2-4

2.3 Logging into the System 2-4
Choosing a Login Name 2-5
The Login Process 2-5
Selecting a Password 2-7
Changing Your Password During Login 2-8
Changing Your Password: **passwd** 2-9

2.4 Entering Commands 2-10
Displaying the Date and Time: **date** 2-10
Displaying User Information: **who** 2-11

2.5 Logging Off 2-11

Chapter 2
Getting Started

2.1 Introduction

The information in this chapter can help you begin using the operating system. It first describes the terminal, then introduces the following steps involved in a typical work session:

- Access the operating system.

  Before you can use the computer, you must identify yourself to the operating system. This step is called *logging in.*

- Enter commands.

  When you have logged in successfully, you can execute commands and run programs.

- Leave the operating system.

  When you finish working with the computer, you must *log out* to end the work session.

2.2 The Terminal

A terminal is an input/output device. You use it to enter requests to the operating system, and the system uses it to send responses to you. There are two basic types of terminals: video display terminals and printing terminals.

A video display terminal shows input and output on a display screen. A printing terminal shows input and output on continuously-fed paper. In most cases, this difference has no effect on the user's actions or the system's responses. Throughout this book, instructions that refer to the terminal screen also apply to the paper in a printing terminal.

2.2.1 Keyboard Characteristics

Although there is no standard layout for terminal keyboards, most keyboards are similar to a typewriter. However, terminal keyboards also include additional keys for functions such as interrupting tasks. These keys are labeled with characters or symbols that identify their functions. Their placement can vary from terminal to terminal.

While the terminal keys are labeled with characters that are meaningful to you (such as the letters of the alphabet), each key is also associated with a code that is meaningful to the computer.

2.2.2 Typing Speed

When you are working at the terminal you can type as fast as you want, even when the operating system is executing a command. The operating system has read-ahead capability that allows it to handle input and output separately. The system receives and stores input (your next request) while it sends output (its response to your last request) to the screen.

2.2.3 Correcting Typing Errors

Your keyboard includes two keys for correcting typing errors: an erase key and a kill key. By default, [#] erases the last character typed and [@] kills (or deletes) the current line. However, you can reassign the erase and kill functions to other keys. Your system administrator may have reassigned one or both of these keys for you.

Deleting the Current Line

[@] deletes the current line. When you press it, an "at" symbol (@) appears at the end of the line and the cursor moves to the next line. Although the line containing the error is still on the screen, the system ignores it.

[@] works only on the current line. If you want to kill a line, be sure to press [@] before you press [Return]. In the following example, we misspelled the **who** command. We used [@] to cancel the line and then retyped the command.

```
$ whooo @
who [Return]
```

Deleting Characters

[#] deletes the last character typed on the current line. When you press [#], a pound sign (#) appears on the screen. This character cancels the preceding character (in effect, erasing it).

You can continue pressing [#] to delete as many characters as you want. For example, in the following command line, we typed [#] twice to delete two characters.

```
$ dattw##e [Return]
```

The operating system interprets this line as the **date** command, typed correctly.

If [#] does not delete characters on your terminal, it is possible that your system administrator moved this delete function to [BackSpace], as described in the next paragraph.

Using BackSpace to Delete Characters. Many people prefer to use [BackSpace] for the erase function. When you press [BackSpace], the cursor moves from right to left, deleting each character it crosses. The characters being deleted remain on the screen until you either type over them or press [Return].

[#] and [BackSpace] are equally effective at deleting characters, but [BackSpace] gives you better visual information about what you are doing.

NOTE

If you don't know what the delete keys are on your terminal, refer to "Setting Terminal Options: **stty***" in Chapter 4. This section tells how to display or change the current delete keys.*

Using a Delete Character as Text

If you are using [#] or [@] as a delete key and you want to use one of these characters as a unit of text either on the command line or in a file, you must tell the operating system to ignore the delete function attached to the character. To do this, type a backslash (\) before the character.

For example, suppose that [#] is your erase key and you want to add the following sentence to a file:

Only one # appears on this sheet of music.

To prevent the operating system from interpreting [#] as a request to delete a character, type a backslash (\) in front of it:

Only one \# appears on this sheet of music.

If you don't include the backslash, the system will erase the space after the word one and print your sentence as follows:

```
Only one appears on this sheet of music.
```

2.2.4 Control Characters

The control key, in combination with other characters, is used to issue commands. This key might be labeled [Control] or [Ctrl] and is probably located to the left of [A] or below [Z].

Commands entered with [Ctrl] are called control characters. They perform tasks such as backspacing and tabbing, and they define commands that are specific to the operating system. For example, [Ctrl-S] temporarily halts the printing of output on a terminal screen; [Ctrl-Q] restarts the output.

In addition, the operating system uses control characters to provide capabilities that are not available on some terminals. For example, if your keyboard does not have a backspace key, you can use [Ctrl-H] instead.

To type a control character, hold down [Ctrl] and press the appropriate alphabetic key. You do not need to press the [Shift] key. Most control characters do not appear on the screen when you type them.

2.3 Logging into the System

The operating system stores information about each user authorized to work on the system. This information is called your login account. It includes the name you use to identify yourself to the system (your login name) and a password. Your system administrator will either assign you a login name or ask you to select your own. You will probably also be assigned a temporary password that you can change later.

2.3.1 Choosing a Login Name

If you are selecting your own login name, keep these requirements in mind:

- A login name can be 1 to 8 characters long.
- A login name can contain any combination of lowercase alphanumeric characters, but must start with a letter.
- A login name cannot contain symbols.

starship, *mary2*, and *jmrs* are examples of legal login names.

2.3.2 The Login Process

When you want to work with the computer, you must first log into the operating system. This step involves typing your login name and password, which the system uses to verify that you are an authorized user.

When you log in, you will probably be using either a terminal wired directly to the computer or a terminal that communicates with the computer over a telephone line. There are many ways to log in over a telephone line. For example, security precautions on your system might require that you use a special telephone number or other security code. For instructions on logging in over a telephone line, see your system administrator.

If your terminal is connected directly to the computer, the `login:` prompt appears in the upper lefthand corner of the screen when you turn your terminal on. Type your login name at this prompt and press Return. For example, if your login name is *starship*, your login line will look like this:

```
login: starship Return
```

Remember to type lowercase letters. If you use uppercase letters when you log in, the operating system will assume that your terminal cannot generate lowercase letters. It will then display only uppercase letters during the login session. The operating system will accept and run many commands typed in uppercase, but will not allow you to use the text editor programs.

Next, the system prompts you for your password. Type your password and press Return. For security reasons, the operating system does not print (or echo) your password on the screen. (When you log in for the first time, the system might require you to change your password before you can complete the login process. The next two sections tell how to select a new password and to change your password while you are logging in.)

If both your login name and password are valid, the system will print the shell command prompt. If you are using either the Bourne or Korn shell, you will see a dollar sign ($). If you are using the C shell, you will see a percent sign (%).

(Most of the examples in this manual assume the Bourne shell is being used and show the dollar sign prompt ($). However, unless noted otherwise, the commands illustrated by the examples work the same way with either the Korn or C shell.)

The system might also display the message of the day, which might be a schedule for system backups. A list of current news items might also appear. You can use the **news** command described in Chapter 10 to read these items. When you are logged in, your screen will resemble this one:

```
login: starship Return
password:          Return
BACKUPS: Sunday, 08/14/88
$
```

If you make a typing mistake when logging in, the operating system prints the message `login incorrect` on your screen. Then it gives you a second chance to log in by printing another `login:` prompt.

```
login: ttarship Return
password:       Return
login incorrect
login:
```

If you experience any problems when logging in, contact your system administrator.

Other Information

login(1)

2.3.3 Selecting a Password

Passwords are an important part of system security. Be sure to keep your password private, to change it frequently, and to choose passwords that are not everyday words. When you select a new password, keep the following requirements in mind:

- Each password must have at least 6 characters. If a password has more than 8 characters, only the first 8 characters are significant.
- Each password must contain at least 2 alphabetic characters and at least 1 numeric or special character. Alphabetic characters can be either uppercase or lowercase letters.
- Each password must differ from your login name and any reverse or circular shift of that login name. For comparison purposes, an uppercase letter and the corresponding lowercase letter are equivalent.
- A new password must differ from the old password by at least 3 characters. For comparison purposes, an uppercase letter and the corresponding lowercase letter are equivalent.

Examples of valid passwords are mar84ch, J0nath0n, and BRAV3S. Your site might have different requirements to consider when choosing a password. Ask your system administrator for details.

2.3.4 Changing Your Password During Login

To maintain system security, many sites use password aging, which causes your password to expire periodically. When your password expires, the system will require you to change your password before you can log in. Password aging is also used to require new users to choose a password before logging in for the first time.

The administrative procedures at your site will determine the steps you will follow to change your password during the login process. However, your procedure will probably be similar to this example:

1. The operating system displays the `login:` prompt. Type your login name and press Return.
2. The operating system prints the password prompt. Type your password and press Return.
3. The system tells you your password has expired and that you must select a new password.
4. The system asks you to type your old password again. Type your password and press Return.
5. The system prompts you to type your new password. Type the password you have chosen and press Return. The system now verifies that the password meets the construction requirements outlined in the previous section.
6. For verification, the system asks you to re-enter your new password. Type your new password again and press Return.

   If you do not re-enter the new password exactly as typed the first time, the system tells you the passwords do not match and asks you to try the procedure again. (At some sites, the communication link is dropped if you do not re-enter the password exactly as typed the first time. If this happens, return to step 1 and begin the login procedure again.) When the passwords match, the shell prompt (`$` or `%`) will appear.

The following screen summarizes this procedure.

```
login: starship Return
password:       Return
Your password has expired.
Choose a new one.
Old password:       Return
New password:       Return
Re-enter new password:       Return
$
```

2.3.5 Changing Your Password: passwd

The **passwd** command allows you to select a new password after you are logged in. To use this command, type **passwd** and press Return. The system will then prompt you to type your old password, then to type your new password twice, as shown on the following screen.

```
$ passwd Return
Old password:       Return
New password:       Return
Re-enter new password:       Return
$
```

When you enter the new password the first time, the system will verify that it meets the construction requirements described earlier in this chapter. When you enter the password the second time, the system verifies that the two copies of the password are identical. If they are not identical, the system will prompt you to enter the new password again.

Other Information

passwd(1)

2.4 Entering Commands

When the shell prompt (`$` or `%`) appears on your terminal screen, the operating system is waiting for you to issue a command. Chapter 1 described the syntax of an operating system command line. This section introduces a few simple commands that you can try now.

If you want to stop the execution of a command, press the interrupt key, which is typically Delete or Break. The operating system will then stop the program and print a shell prompt (`$` or `%`) on the screen.

2.4.1 Displaying the Date and Time: date

The **date** command prints the date and time, using the 24-hour clock. At the shell command prompt, type **date** and press Return. The operating system then accesses a program called **date**, executes it, and prints its results on the screen, as shown below.

```
$ date Return
Wed Oct 15 09:49:44 EDT 1986
$
```

Other Information

date(1)

2.4.2 Displaying User Information: who

The **who** command lists the login names of everyone currently working on your system. To use this command, type **who** and press Return. The following screen shows sample output from the **who** command.

```
$ who Return
starship     ttyGA/GABB  Oct 12   8:53
mary2        ttyGA/GABD  Oct 12   8:56
jmrs         ttyGA/GABE  Oct 12   8:56
$
```

The tty designations refer to the special files corresponding to each user's terminal. The date and time that each user logged in also appears.

Other Information

who(1)

2.5 Logging Off

When you finish working on the computer, you must log off of the system. The procedure you will follow depends on the shell program you are using.

- Bourne and Korn shells

  To log out, type Ctrl-D after the shell prompt (`$`). (To type this control character, hold down Ctrl and press D. The characters do not appear on your screen.) The operating system will then display the `login:` prompt, showing that you have logged off successfully and the system is ready for someone else to log in.

```
$ Ctrl-D
login:
```

The **exit** command also allows you to log off of the system. If you are using the Korn shell and have set the **ignoreeof** option, you must use **exit** to log out. This command is also used in shell programs.

- C shell

  To log out, type **logout** after the shell prompt (`%`). The operating system will then display the `login:` prompt, showing that you have logged off successfully and the system is ready for someone else to log in.

```
% logout  Return
login:
```

NOTE

Always log off of the operating system before you turn off the terminal or hang up the telephone. If you don't take this step, you might not be logged off of the system.

Other Information

csh(1), **ksh**(1), **sh**(1)

Chapter 3
Using the Filesystem

3.1 **Introduction** 3-1
The Filesystem Structure 3-1
Your Home Directory 3-1
Your Current Directory 3-3
Pathnames 3-3
Full Pathnames 3-3
Relative Pathnames 3-4
Naming Directories and Files 3-6
File and Directory Access 3-7
What Permissions Mean 3-9
The Superuser 3-10

3.2 **Managing Directories and Files** 3-11
Redirecting Output 3-11
Creating Directories: **mkdir** 3-12
Changing Your Current Directory: **cd** 3-13
Displaying Your Current Directory: **pwd** 3-14
Removing Directories: **rmdir** 3-15
Listing the Contents of a Directory: **ls** 3-16
Listing All Names in a Directory 3-17
Listing Contents in Short Format and Adding Labels 3-18
Listing Contents in Long Format 3-18
Saving or Printing a Directory Listing 3-20
Creating and Displaying a File: **cat** 3-21
Paging Through a File: **pg** 3-23
Paginating and Displaying a File: **pr** 3-25
Displaying the End of a File: **tail** 3-27
Copying a File: **cp** 3-28
Moving or Renaming a File: **mv** 3-30
Linking a File: **ln** 3-31
Removing a File: **rm** 3-33
Changing File and Directory Permissions: **chmod** 3-35
Changing Permissions with Symbols 3-35
Changing Permissions with Octal Notation 3-36
Counting Lines, Words, and Characters in a File: **wc** 3-38

Identifying Differences Between Files: **diff** 3-40
Searching a File for a Pattern: **grep** 3-42
Creating a Regular Expression 3-43
Locating Files: **find** 3-46
Sorting and Merging Files: **sort** 3-49
Changing the Sort Order 3-51
Sending Sorted Output to a File 3-51
Sorting on Fields 3-52
Cutting Fields or Characters from a File: **cut** 3-56

3.3 Changing to Another Group: newgrp 3-58

3.4 Disk Quotas 3-59
Limits 3-59
Checking Quotas and Limits 3-60
Responding After Reaching a Hard Limit 3-61
Quotas and ptx/NFS 3-61

Figures

3-1 Home directories in the filesystem 3-2
3-2 The *starship* directory 3-5
3-3 Output from the **ls –l** command 3-19

Chapter 3
Using the Filesystem

3.1 Introduction

This chapter introduces several commands that allow you to work in the filesystem. You can use these commands to create, copy, move, and delete directories and files, to read the contents of files, and to locate information within a file.

Before you begin using the commands, you should be familiar with the structure of the filesystem and your location within it, and you should know how to specify file and directory names. The remainder of this section describes these concepts.

3.1.1 The Filesystem Structure

As described in Chapter 1, the structure of a filesystem resembles an upside-down tree. The filesystem begins at a directory called root that is represented by a slash (/). The level below the root directory contains several system directories such as */bin* and */usr*. When you move down from these directories, you can reach all of the directories and files in the filesystem. Figure 3-1 shows a sample filesystem.

In the filesystem hierarchy, a directory and the files and directories beneath it have what is called a parent/child relationship. This type of relationship is possible for many layers of files and directories. There is no limit to the number of files and directories you can create within a directory, allowing you to organize your files in a variety of ways.

3.1.2 Your Home Directory

When you complete the login procedure, the operating system places you at a specific point in the filesystem structure called your home directory. Your login name is usually the name of your home directory.

Typically home directories are grouped together in one or more system directories. For example, in the sample filesystem shown in Figure 3-1, a system directory called *user1* contains the home directories for login names *starship*, *mary2*, and *jmrs*. The *user1* directory is on the same level as the system directories discussed in Chapter 1.

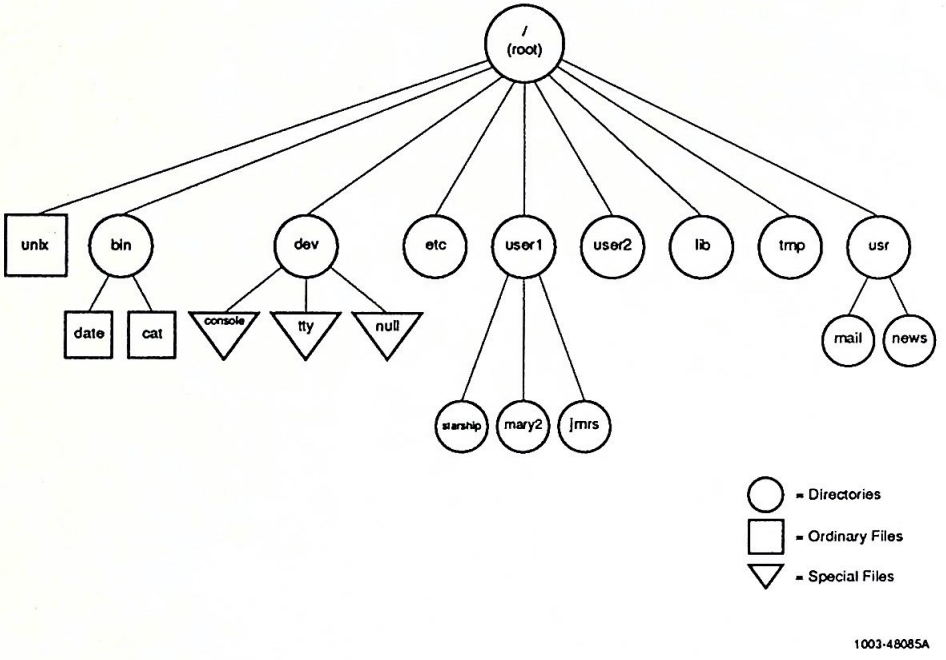


Figure 3-1. Home directories in the filesystem. This filesystem includes three home directories: *starship*, *mary2*, and *jmrs*.

You own the contents of your home directory. In our example, user *starship* owns the *starship* directory, user *mary2* owns the *mary2* directory, and user *jmrs* owns the *jmrs* directory.

When you own a directory, you control access to it. You determine who can create files and additional directories (sometimes called subdirectories) within that directory, and you determine who can read, move, or delete files and directories. Similarly, when you own a file, you determine who can read or change its contents. If the file contains a program, you determine who can execute it.

You have full responsibility for everything in your home directory. It is a vantage point for viewing both your files and directories and the rest of the filesystem.

3.1.3 Your Current Directory

The directory you are working in is called your current directory (or current working directory). When you are working in your home directory, it is considered your current directory. If you move to another directory, that directory becomes your current directory.

You can use the **pwd** command (described later in this chapter) to print the name of your current directory. This command is helpful when you are moving frequently from one directory to another and need to know your current location to execute a command properly.

3.1.4 Pathnames

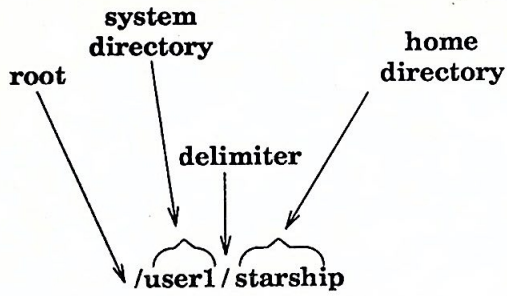
A pathname identifies the location of each file and directory in the filesystem. You can identify a file or directory in two ways: by a full pathname or by a relative pathname.

Full Pathnames

A full pathname (also called an absolute pathname) starts at the root directory and lists all directories in the filesystem structure down to a particular directory or file. You can use a full pathname to reach any file or directory in the filesystem.

Because a full pathname starts at the root directory, its leading character is always a slash (/), the character representing the root directory. The final name in a full pathname can be either a file or a directory name. All other names in the path must be directories. The names are separated by slashes (/). These slashes have no special meaning; they are simply delimiters.

For example, if you are working in the *starship* directory located in *user1*, the full pathname of your current directory is */user1/starship*.



/	= The root directory of the filesystem
user1	= The system directory one level below root in the hierarchy
/	= A delimiter separating the directory names *user1* and *starship*
starship	= The current working directory

The portion of the pathname preceding the actual filename is called the path prefix. For */user1/starship*, the path prefix is */user1/*.

Relative Pathnames

A relative pathname starts in your current directory and leads up or down through a series of directories to a particular file or directory. If you move down from your current directory, you can access the files and subdirectories in your current directory. If you move up from your current directory, you can pass through layers of parent directories to the root directory. You can also move up to a parent directory and then descend to other directories and files.

A relative pathname begins with one of the following:

- A directory or filename
- A dot (.), which represents your current directory
- A double dot (..), which represents the directory immediately above your current directory (the parent directory of your current directory)

The relative path to a directory or file in your current directory is simply the name of the file or directory. For example, if your current directory is *starship*, the relative pathname to its *draft* subdirectory is *draft*. Figure 3-2 shows the contents of the *starship* directory in our sample filesystem.

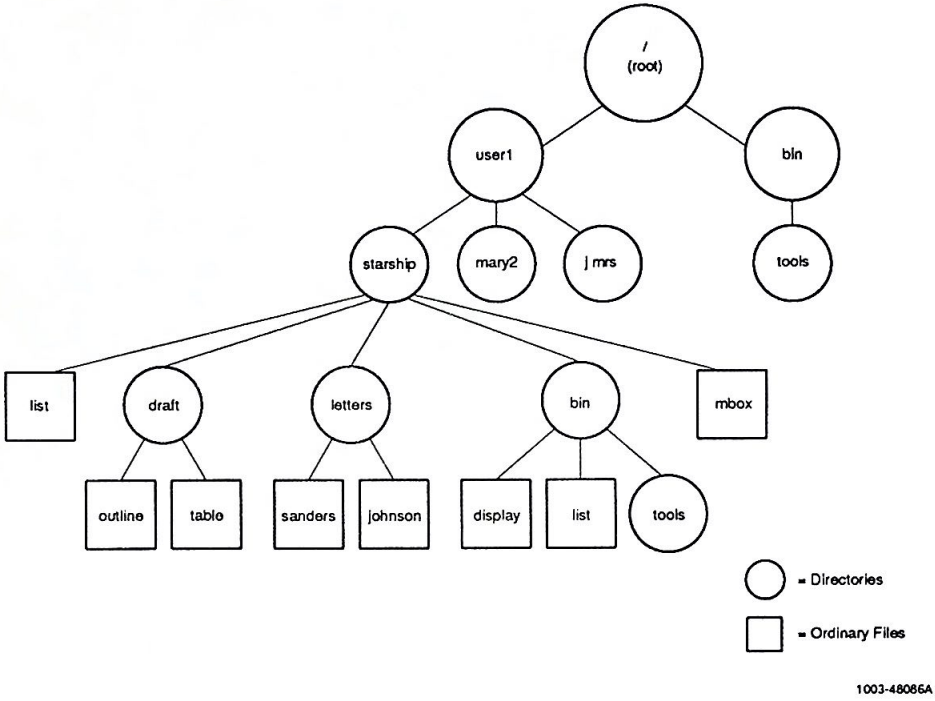


Figure 3-2. The *starship* directory. The bold line traces the relative path from the *starship* directory to the *draft* directory.

The *draft* directory contains two files: *outline* and *table*. The relative pathname from *starship* to *outline* is *draft/outline*.

You can also move up the filesystem structure and then descend into other files and directories. To move to the parent of your current directory, use the .. notation. For example, if you are in the *draft* directory, .. is the pathname to the *starship* directory, and ../.. is the pathname to *user1*, *starship*'s parent directory.

From the *draft* directory, you can trace a path to the *sanders* file in the *letters* directory. The pathname is *../letters/sanders*. The .. brings you up to *starship*. Then, the names *letters* and *sanders* take you down through the *letters* directory to the *sanders* file.

Keep in mind that you can always use a full pathname in place of a relative pathname.

Following are some examples of full and relative pathnames.

/	Full pathname of the *root* directory
/bin	Full pathname of the *bin* directory (contains most executable programs and utilities)
/user1/starship/bin/tools	Full pathname of the *tools* directory
bin/tools	Relative pathname of the file or directory *tools* in the directory *bin* If the current directory is /, the equivalent full pathname is */bin/tools*. If the current directory is *starship*, the equivalent full pathname is */user1/starship/bin/tools*.
tools	Relative pathname of the *tools* file or directory located in the current directory

3.1.5 Naming Directories and Files

You can give your directories and files any names you want, as long as you observe the following rules:

- A name can be from 1 to 255 characters long.
- All characters other than a slash (/) are legal.

To avoid confusion when the shell interprets your commands, we recommend that you also follow these guidelines when naming directories or files:

- Avoid using the following characters, which have special meaning to the operating system. They are used on the command line to tell the shell to perform tasks such as redirecting command output to a file. (Chapter 4 describes these characters.)

  ? @ # $ ^ & * ( ) ` [ ] \ | ; ' " < >

- Avoid using a space, tab, or backspace in directory or filenames. If you use a space or tab, you must enclose the name in quotation marks when you type it on the command line.
- Avoid using a plus sign (+), a minus sign (–), or a period (.) as the first character in a filename.

The operating system distinguishes between uppercase and lowercase letters. For example, the system considers a directory named *draft* to be different from one named *DRAFT*.

Following are examples of legal names for directories and files:

memo	*MEMO*	*section2*	*ref:list*	*test_two*
file.d	*chap3+4*	*item1-10*	*outline*	*Part4*

When a filename includes a period, such as *file.d*, the portion of the filename following the period is called a filename extension or suffix. You can use as many suffixes as necessary to make your filenames easy to understand.

Some programs expect specific filename suffixes. For example, a file ending in *.c* is a C program file, a file ending in *.f* is a FORTRAN program file, and a file ending in *.p* is a Pascal program file. Files ending in *.o* are object files.

3.1.6 File and Directory Access

When you create a directory or file, the system assigns a set of permissions that determine who can read, write, and execute that file or directory. Similarly, the permissions set on other users' files and directories determine if you can read, write, or execute them.

The **ls –l** command (described later in this chapter) shows the permissions assigned to a file or a directory. For example, **ls –l** produces the following output for the */user1/starship/bin* directory shown in Figure 3-2.

```
$ ls -l  Return
total 35
-rwxr-xr-x  1 starship  project   9346  Nov 1  08:06  display
-rw-r--r--  1 starship  project   6428  Dec 2  10:24  list
drwx--x--x  2 starship  project     32  Nov 8  15:32  tools
$
```

This command prints a line for each file or subdirectory in the current directory. The directory or filename appears at the end of the line.

The first character of the line tells you the file type. A hyphen (–) represents a regular file; a `d` represents a directory. The **ls –l** output shows the *bin* directory contains two files, *display* and *list*, and one directory, *tools*.

The remaining characters specify the permissions assigned to the file or directory. Permissions are assigned for three categories of users: the owner, the group, and all other users. These categories provide security for your files and directories.

The owner is usually the person who created the file or directory. For example, you own all of the files and subdirectories you create within your home directory. If you want to keep a file private, you can set the permissions so that only you can access it.

The group category defines permissions for members of the owner's group. When your login account was set up, you were assigned to a specific group of users. Each site can create its own grouping scheme. At some sites, all users belong to a single group. At other sites, users are assigned to different groups based, for example, on their department or job function. If members of your group need to work with your files, you can set the permissions to allow them to read, write, or execute the files.

The last category, other users, refers to all users who are not members of the owner's group. This category allows you to permit or restrict access to your files for all other users on your system. The permissions for this category are also called "world access."

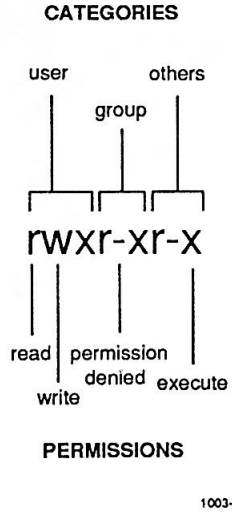
For the */user1/starship/bin* directory, the permissions are as follows:

`rwxr-xr-x` (the *display* file)
`rw-r--r--` (the *list* file))
`rwx--x---` (the *tools* directory)

The first three characters define permissions for the owner, the next three characters define permissions for the owner's group, and the last three characters define permissions for all other users.

Within each set of characters, an `r` indicates that read permission is allowed, a `w` indicates that write permission is allowed, and an `x` indicates that execute permission is allowed. If a dash appears in place of an `r`, `w`, or `x`, that permission is denied.

The following diagram summarizes this breakdown for the *display* file.



As you can see, the owner has `r`, `w`, and `x` permissions. Members of the owner's group and all other users have `r` and `x` permissions only.

What Permissions Mean

Permissions have different effects on files and directories. For a file, read permission allows the user to invoke a command such as **cat** or **pg** to read the contents of a file. Write permission allows the user to change the file. For example, the user can invoke a text editor to edit the file. Execute permission applies only to files containing programs. It allows the user to execute, or run, the program.

For a directory, read permission allows the user to run the **ls** command to list the contents of the directory. Write permission allows the user to create new files in the directory and to remove existing files. Execute permission allows the user to invoke the **cd** command to make that directory the user's current directory.

To execute many commands affecting a file, you must have specific permissions on the directory containing that file. For example, to issue a command to read a file, you must have execute permission on the directory containing that file. To use an editor program to change a file, you must have write permission on the directory containing that file.

If you are working with files belonging to another user and you are unable to execute a command, you probably don't have the file or directory permissions the command requires.

The Superuser

To maintain system security, access to many system files and to the commands that affect those files is limited to a privileged user called the *superuser*. Because the superuser uses **root** as a login name, the superuser is also called "root."

The superuser performs activities such as starting up or shutting down the system, creating new login accounts, changing passwords, and other administrative tasks. The superuser has access to all files, regardless of their permissions.

Other Information

"Listing the Contents of a Directory: **ls**" and "Changing File and Directory Permissions: **chmod**" later in this chapter

chmod(1), **ls**(1)

3.2 Managing Directories and Files

To work in the filesystem, you must be able to create, delete, copy, and move files and directories, to read the contents of files, to specify the users allowed access to your files and directories, and to locate information within a file. The operating system provides shell commands to perform these tasks. This chapter describes several of these commands. To help you become familiar with them, we suggest that you try out the commands as you read about them.

When using these commands, you will frequently need to specify the file or directory the command is to affect. When you type a file or directory name, keep in mind that you can specify either a full or a relative pathname.

Many of these commands include options that modify how the command works. The descriptions in this chapter include only the commonly used options. For a complete list of options, refer to the page in the *Reference Manual* listed at the end of each command description.

You can also use the **assist** feature described in Chapter 5 to help you create a complex command line.

3.2.1 Redirecting Output

When you execute many commands, their output appears on your screen. Occasionally, you might want to save this output or keep it from printing on the screen. You can use the greater-than sign (>) to perform this task, which is called *output redirection*. When you type a command followed by a greater-than sign (>) and the name of a file, the output from the command is sent to the file. If the file already exists, the command output overwrites the original contents of the file. If the file does not exist, it is created.

As an example, the following command line sends the output from the **date** command to the *date.output* file instead of printing it on your screen.

```
$ date > date.output Return
```

Other Information

"Input and Output Redirection" in Chapter 4

3.2.2 Creating Directories: mkdir

The **mkdir** (short for *make directory*) command creates a new directory under the current directory. You will remain in your current directory. If you want to work in the new directory, use the **cd** command (described in the next section) to move to that directory. You can then build additional subdirectories in that directory.

To create a new directory, type **mkdir** followed by the name of the directory. For example, to create a directory called *notices*, you would enter this command:

```
$ mkdir notices Return
```

When the command completes, the shell command prompt appears on your screen.

You can create several directories with a single command. Simply list the directories on the command line, leaving a space between each name. The following command creates three directories: *notices*, *letters*, and *memos*.

```
$ mkdir notices letters memos Return
```

As you begin creating directories, we suggest you develop a scheme that will make it easy to locate information in your files. If you put related files in the same directory, it will be easy to find them later. You can name your new directories anything you want as long as you follow the guidelines listed earlier under "Naming Directories and Files."

Other Information

mkdir(1)

3.2.3 Changing Your Current Directory: cd

When you log into the operating system, you are placed in your home directory. As long as you work there, it is also your current working directory. The **cd** command (short for *change directory*) allows you to move to another directory, which then becomes your current directory.

To use this command, type **cd** followed by the pathname of the directory you are moving to. For example, to move from the *starship* directory shown in Figure 3-2 to its subdirectory *draft*, you would type this command:

```
$ cd draft Return
```

The *draft* directory is now your current directory. You can create subdirectories in it by using the **mkdir** command, and you can creates files by using the **cat** command. You can also use the **ed** or **vi** editors described later in this manual to create and edit files.

You can use any valid pathname (full or relative) as an argument to the **cd** command. For example, to move to the *letters* directory from the *draft* directory, you can specify either the full pathname */user1/starship/letters* or the relative pathname *../letters*. (The .. notation moves you to the *starship* directory; the rest of the pathname moves you to *letters*.) The following command lines show both of these pathnames.

```
$ cd /user1/starship/letters Return

$ cd ../letters Return
```

If you do not specify a pathname, the **cd** command moves you to your home directory.

```
$ cd Return
```

Other Information

cd(1)

3.2.4 Displaying Your Current Directory: pwd

The **pwd** (short for *print working directory*) command prints the name of your current directory. You can use this command whenever you need to verify your location. For example, you might type a command to copy a file and receive a response that the file doesn't exist. To make sure you typed the correct pathname to the file, you can use the **pwd** command to print the name of your current directory.

To use this command, type **pwd** and press Return. For example, if you are working in the *starship* directory shown in Figure 3-2, the **pwd** command will display the full pathname shown on the following screen.

```
$ pwd  Return
/user1/starship
$
```

Other Information

pwd(1)

3.2.5 Removing Directories: rmdir

The **rmdir** command (short for *remove directory*) removes unneeded directories. You cannot delete your current directory; you must be working in another directory. Also, the **rmdir** command can remove only directories that you own (typically, directories that you have created).

To use this command, type **rmdir** followed by one or more directory names. For example, if you are working in your home directory, you could delete the *outline* subdirectory by typing the following command. The shell prompt reappears when the command is complete.

```
$ rmdir outline Return
```

The **rmdir** command removes only empty directories. If you want to delete a directory containing files or subdirectories, you must first delete the files and subdirectories and then delete the directory itself. For example, your home directory might contain a *memos* directory that includes one subdirectory, *tech*, and two files, *june.30* and *july.31*. If you try to remove the *memos* directory (by issuing the **rmdir** command from your home directory), the system prints the following message:

```
$ rmdir memos Return
rmdir:  memos not empty
$
```

Before removing the *memos* directory, you must first delete its contents: the *tech* subdirectory and the *june.30* and *july.31* files. You can use **rmdir** to remove the *tech* subdirectory. (For instructions on removing files, see the description of the **rm** command later in this chapter.) After removing the contents of the *memos* directory, use **rmdir** to delete *memos* itself.

Other Information

rm(1) (includes **rmdir**)

3.2.6 Listing the Contents of a Directory: ls

All directories store information about the files and subdirectories they contain. This information includes the name of each directory or file, its size, and the date it was last modified. The **ls** (short for *list directory*) command will display this information for any directory that you have permission to access.

The **ls** command does not print the contents of a file. If you want to read a file, use the **cat**, **pg**, or **pr** commands. You can also use the the **ed** or **vi** editors, described later in this manual, to read a file.

When you type **ls** and press Return, the system prints, in alphabetical order, the names of files and directories in your current directory. (Names beginning with a number or an uppercase letter print first.) The following screen shows the output **ls** produces for the *starship* directory shown in Figure 3-2.

```
$ ls  Return
bin
draft
letters
list
mbox
$
```

To print the names of files and subdirectories in a directory other than your current directory, specify the name of that directory after the **ls** command. For example, to list the contents of the *draft* directory (a subdirectory of *starship*), you would type this command:

```
$ ls draft  Return
```

To list the contents of *user1* (the parent directory of *starship*), include either its full or relative pathname in the command line, as shown below.

```
$ ls /user1  Return
```

```
$ ls ..  Return
```

The **ls** command can determine if the current directory contains a specific file. To use the command in this manner, type **ls** followed by the name of the file. On the following screen, we used the **ls** command to determine if the *starship* directory contains either the *list* or the *notes* files.

```
$ ls list notes  Return
list
notes not found
$
```

If the current directory contains the file, the system prints the filename. If the current directory does not contain the file, the system will display a message stating the file was not found.

The options to the **ls** command can give you more information about a file or directory. Although there are more than a dozen **ls** options, this section describes only the most commonly used ones.

Listing All Names in a Directory

Some filenames such as *.profile* begin with a period (or dot). These files typically contain startup information used by the operating system. The **ls** command does not include these files in the list it prints on the screen. If you want the output from **ls** to include filenames beginning with a dot, use the **–a** option on the command line.

The screen on the next page shows the output this option produces for *starship*, the current directory. (The . and .. entries refer to the directory itself.)

```
$ ls -a  Return
.
..
.profile
bin
draft
letters
list
mbox
$
```

Listing Contents in Short Format and Adding Labels

The **–C** option lists the contents of a directory in multiple columns rather than in a single column. The **–F** option labels executable files with an asterisk (*) and directories with a slash (/). The command on the following screen uses both of these options.

```
$ ls -CF  Return
bin/          letters/        mbox
draft/        list*
$
```

Listing Contents in Long Format

The **–l** option is the most informative. It prints the contents of a directory in long format, including permissions, number of links, owner, group, size in bytes, and time of last modification for each file or subdirectory. Figure 3-3 shows the output that the **ls –l** command produces for the *starship* directory.

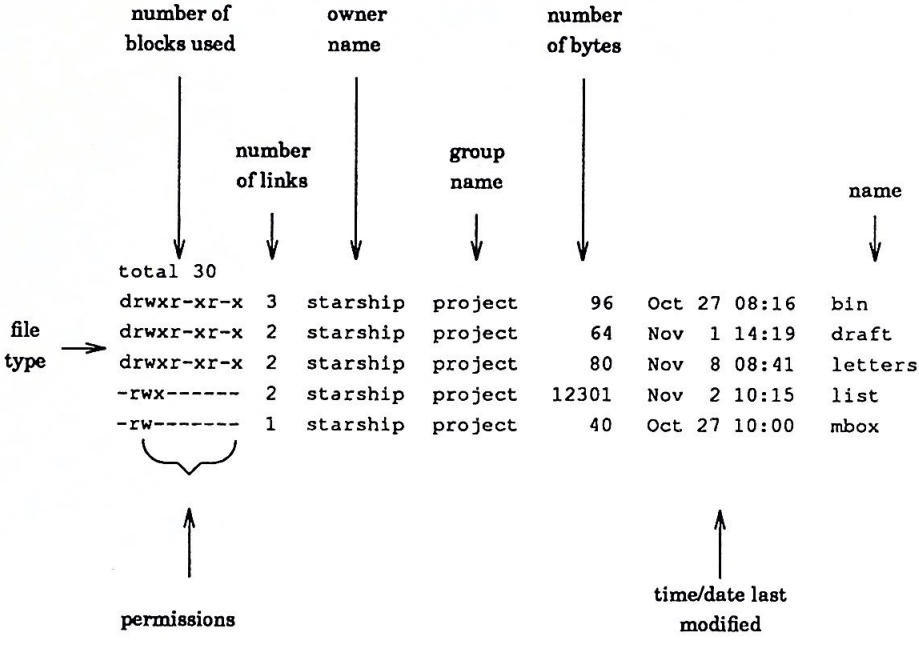


Figure 3-3. Output from the ls –l command.

The output contains this information:

Number of blocks used
: The first line (`total 30`) shows the amount of disk space the directory uses, measured in blocks.

File type
: The first character in each line tells you the type of file: `a d` represents a directory, a hyphen (–) represents an ordinary disk file, and `b` and `c` denote special files, which are devices such as disk drives or terminals. The *starship* directory contains three directories and two ordinary disk files.

Permissions
: This set of characters, which are either letters or hyphens, identify who has permission to read, write, or execute the file or directory.

Number of links
: For a file, this number tells you how many links exist to that file. For a directory, the link count shows the number of directories immediately under it plus two (for the directory itself and its parent directory).

Owner name
: This column contains the login name for the owner of the directory or file (here, it is `starship`).

Group name
: This column contains the group to which the owner belongs (in this example, the `project` group).

Number of bytes
: This number shows the size of the file or directory measured in units of information (or memory) called bytes.

Time/date last modified
: This column lists the month, day, and time the file was last modified.

File or directory name
: This column contains the name of the directory or file.

Saving or Printing a Directory Listing

Occasionally, you might want to save a directory listing. To do this, redirect the output from the **ls** command to a file, as shown in the following example.

```
$ ls -l > dir.list Return
```

If you want a printed copy of the listing, use the **lp** command described in Chapter 6.

Other Information

ls(1)

3.2.7 Creating and Displaying a File: cat

The **cat** command (short for *concatenate*) has three main uses. It can display the contents of one or more files on your screen, it can create files, and it can join (or concatenate) two or more files. (See Chapter 4 for information on concatenating files.) The following discussion tells how to use **cat** to create files and to print the contents of a file on your screen.

To create a file, type a command in this form:

$ **cat >** *file_name*

The greater-than sign (>) redirects the output from the **cat** command. Here it causes the **cat** command to create a file with the name you specify. As an example, to create a file named *memories*, you would type this command:

$ **cat > memories** Return

You can now type as much text as you want, pressing Return at the end of each line. When you finish entering your text, press Return to go to the next line, then type Ctrl-D to terminate the **cat** command. The shell prompt then reappears on your screen.

```
$ cat > memories  Return
Memories so sweet  Return
Of sugar in the bowl  Return
Spooning it so lightly  Return
Upon my cereal  Return
Ctrl-D
$
```

To print the contents of a file on the screen, type **cat** followed by the name of the file. For example, if you are working in the *letters* directory (shown in Figure 3-2) and want to read the *johnson* file, you would type this command:

$ **cat johnson** Return

The **cat** command then prints the contents of the file, which scroll rapidly on your screen. If you are looking at a long file, the first part will disappear before you can read it. To interrupt the display, press Ctrl-S and scrolling will halt. When you are ready to continue, press Ctrl-Q and **cat** will resume scrolling the file.

To print the contents of two or more files, type the names of the files on the command line, leaving a space between each name. For example, to look at the *johnson* and *sanders* files, you would type this command:

```
$ cat johnson sanders Return
```

The **cat** command will display the *johnson* and *sanders* files in that order on your terminal.

You can view the contents of a file in a directory other than your current directory by typing the appropriate pathname to that file. For example, if your current directory is */user1/starship/draft* and you want to read the *sanders* file in the */user1/starship/letters* directory, you could type the full pathname shown below.

```
$ cat /user1/starship/letters/sanders Return
```

You could also use this relative pathname:

```
$ cat ../letters/sanders Return
```

Other Information

"Appending Output to an Existing File" in Chapter 4

cat(1)

3.2.8 Paging Through a File: pg

The **pg** command (short for *page*) prints the contents of a file page-by-page. It allows you to move forward or backward through the file and to search for specific character patterns. Because **pg** lets you control the display, it is useful for reading long files.

To use this command, type **pg** followed by one or more filenames. For example, to look at the contents of the *outline* file, you would type this command:

```
$ pg outline Return
```

The first page of the file will appear on the screen, followed by a colon prompt (:). When you are ready to read the next page, press Return and **pg** will print the next page of the file. When you reach the end of the file, the message (EOF): appears at the bottom of the screen. When you finish reading the file, press Return and the shell prompt will reappear on your terminal.

While you are using **pg** to examine a file, you can enter any of the following commands at the colon prompt (:). These commands allow you to specify the page that appears on the screen and to search for character patterns.

h	Prints a list of **pg** commands
q or **Q**	Quits **pg**
Return	Shows next page of text
l	Shows next line of text
d or Ctrl-D	Shows next half-page of text
$	Shows last page of text
. or Ctrl-L	Shows current page of text again
f	Skips next page of text and shows following page
n	Shows next file specified on command line

p	Shows previous file specified on command line
w	Sets window size and then shows next page
s *filename*	Saves currently displayed file under *filename*
! *command*	Executes specified command
/pattern	Searches forward for specified character pattern (**pg** does not search the current page)
?*pattern*	Searches backward for specified character pattern (**pg** does not search the current page)

You can preface many of these commands with a number specifying a particular line or page to print on the screen. For example, +1 Return will display the next page, –1 Return will display the previous page, and 1 Return will display the first page of the text.

When you specify a pattern for the */pattern* and **?***pattern* commands, you can use special characters to abbreviate the search pattern or to cause it to match more than one character string. The discussion of the **grep** command, later in this chapter, describes these special characters.

Other Information

pg(1)

3.2.9 Paginating and Displaying a File: pr

The **pr** command will paginate and display the contents of a file on your screen. If you type the **pr** command without any options, it produces output in a single column containing 66 lines per page. A five-line header (two blank lines, a line containing the date, time, filename, and page number, and two more blank lines) precedes the text. Five blank lines print after the text.

The following screen shows sample output from the **pr** command.

```
$ pr johnson  Return

Mar  5 15:43 1986 johnson Page 1

March 5, 1986

Mr. Ron Johnson
Layton Printing
52 Hudson Street
New York, N.Y.

Dear Mr. Johnson:

I enjoyed speaking with you this morning about
your company's plans to automate your business.
Enclosed please find the material you requested
about our office automation software.

Please call if I can be of further assistance to you.

John Howe
.
.
$
```

The ellipses after the last line in the file represent the blank lines that **pr** formats into the output to make the page contain 66 lines.

If you are working on a display terminal, an entire page (66 lines) will scroll on your screen, making it impossible for you to read the first lines in the file unless you have the ability to roll back a screen or two. However, if the file you are examining is particularly long, you still might not be able to read the file. In such cases, type Ctrl-S to interrupt the flow of printing on your screen. When you are ready to continue, type Ctrl-Q to resume printing.

The options to **pr** allow you to change the way **pr** will display a file. For example, you can specify multiple columns, double-spacing, different page numbering, and whether the header should print.

You can also direct the output of **pr** to a different file and then use **pg** to look at the output. To do this with the *forecast* file, you would type these commands:

```
$ pr forecast > newfile Return
$ pg newfile Return
```

You can also send a file that **pr** has paginated to your printer. For more information, see Chapter 6, "Using the Printer."

Other Information

pr(1)

3.2.10 Displaying the End of a File: tail

The **tail** command prints the last part of a file on your screen. If you enter **tail** followed by a filename, **tail** prints the last 10 lines in the file. For example, the following command prints the last 10 lines of the *conference* file.

```
$ tail conference Return
```

If you want the display to start at a different line in the file, follow the **tail** command with a number indicating where the display should begin. You can specify that the display start a certain number of lines from the beginning of the file by preceding the number with a plus sign (+). For example, to look at the *conference* file starting at line 20, you would type this command:

```
$ tail +20 conference Return
```

If you want the display to start a certain number of lines from the end of the file, precede the number with a minus sign (–). To show only the last 15 lines of the file, you would type this command:

```
$ tail -15 conference Return
```

In these examples, the **tail** command printed a specific number of lines. The **tail** command can also display a number of blocks or characters. The **-b** option will display blocks; the **-c** option will display characters. For example, to look at the last 20 characters in the conference file, you would type this command:

```
$ tail +20 -c conference Return
```

Other Information

tail(1)

3.2.11 Copying a File: cp

The **cp** command (short for *copy*) makes a copy of a file and leaves the original file intact. You can place the copy either in the directory containing the original file or in another directory. If the copy of the file will be in the same directory as the original file, you must give the copy a new name.

To copy a file within a directory, type **cp** followed by the filename, then type the name to be assigned to the copy. For example, to make a copy of the *outline* file and name the copy *new.outline*, you would type the following command:

```
$ cp outline new.outline Return
```

You can use the **ls** command to verify the existence of the new file, as shown on the following screen.

```
$ cp outline new.outline  Return
$ ls  Return
new.outline
outline
table
$
```

When you select a name for the copy, be sure the directory does not contain a file with that name. If the file already exists, it will be overwritten by the copy. Using the example on the previous screen, if the current directory already contained a file called *new.outline*, the copy of the *outline* file would overwrite it.

If you are placing the copy in another directory, it is not necessary to give the copy a new name. The operating system allows you to have two files with the same name as long as they are in different directories. If you do not include a name for the copy when you type **cp**, it assigns your new file the same name as the original file.

For example, if you are working in the *draft* directory shown in Figure 3-2, you can copy your *outline* file to the *letters* directory without changing the name of the file. To do this, use either of the following commands. The first command specifies the full pathname; the second command uses a relative pathname.

```
$ cp outline /user1/starship/letters Return
```

```
$ cp outline ../letters Return
```

If you are working in the *letters* directory, you can use the following commands to copy the *outline* file from the **draft** directory. The first command specifies the full pathname; the second command specifies a relative pathname. The dot (.) represents the current directory (the *draft* directory in this example), which is the destination for the copy.

```
$ cp /user1/starship/draft/outline . Return

$ cp ../draft/outline . Return
```

In these examples, **cp** makes a copy of the *letters/outline* file, places the copy in the *draft* directory, and names the copy *outline*.

If you want to give your new file a different name, you must specify it on the command line. For example, to give the copy of the *outline* file the name *plan*, you could use this command from the *draft* directory:

```
$ cp outline ../letters/plan Return
```

From the *letters* directory, you could enter this command:

```
$ cp ../draft/outline plan Return
```

When selecting filenames, keep in mind the naming conventions described in "Naming Directories and Files" earlier in this chapter.

Other Information

cp(1)

3.2.12 Moving or Renaming a File: mv

The **mv** command (short for *move*) renames or moves a file. You can use this command to rename a file within a directory or to move a file to another directory. If you use this command to move a file to a different directory, you can rename the file or retain its original name.

To rename a file within a directory, type **mv** followed by the original filename and then the new filename. For example, to rename the *jackson* file *jackson.88*, you would enter this command:

$ **mv jackson jackson.88** Return

When the command completes, the shell prompt appears. If you want to verify the new filename, use the **ls** command to list the contents of the directory. You can also rename a directory in this way.

When you use **mv** to move a file from one directory to another, you can keep the same filename or change it to a different name. To move the file without changing its name, type a command in this form:

$ **mv** *file directory*

For example, you might want to move the *table* file from the current directory (*/user1/starship/draft*) to the */user1/starship/letters* directory. You can use either of these commands to move the file:

$ **mv table /user1/starship/letters** Return

$ **mv table ../letters** Return

If you want to rename the file when you move it, add the new name to the pathname of the destination directory. For example, you could use one of the following commands to move the *table* file and give it the name *chart*.

$ **mv table /user1/starship/letters/chart** Return

$ **mv table ../letters/chart** Return

Other Information

cp(1) (includes **mv**)

3.2.13 Linking a File: ln

The operating system uses pointers to associate a filename with the place the file is stored on disk. These pointers are called links. When you create a file, it has one link. You can create additional links to allow users to share a file or to make a file appear in more than one directory within a large directory structure.

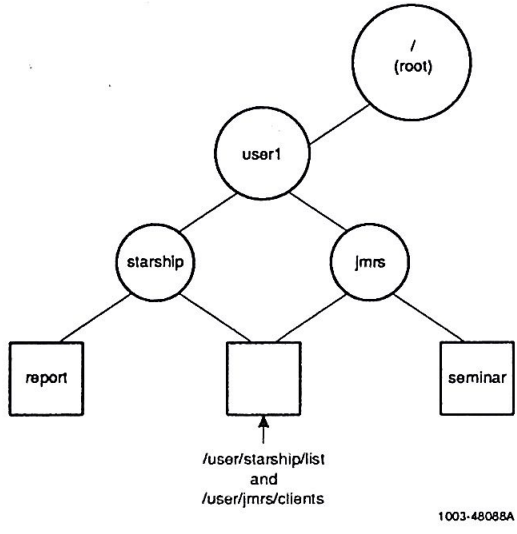
The **ln** (short for *link*) command creates a link to an existing file. It does not copy the file. Instead, it creates an additional pointer to the location of the file on disk. The link appears as a filename in the directory you specify. The new link can have the original filename or you can assign a new name.

To create a link to a file, type **ln** followed by the name of the file and the pathname to the link. For example, if you are working in */user1/starship*, you could create a link for your *list* file so that it appears in the *jmrs* directory under the name *clients*. To do this, you would enter the following command:

```
$ ln list /user1/jmrs/clients Return
```

For this command to work, you must have write permission on the *jmrs* directory. The owner of the directory can use the **chmod** command, described later in this chapter, to give you write permission.

When the **ln** command completes, the file can be accessed with either of these pathnames: */user1/starship/list* and */user1/jmrs/clients*. The diagram on the next page shows what the directory structure looks like.



The operating system treats all links to a file equally. It cannot tell which link was created first. If a file has more than one link, you can delete a link (by using the **rm** command described in the next section) and still have access to the file through the remaining links. To delete the file itself, you must delete all of the links.

Other Information

cp (includes **ln**)

3.2.14 Removing a File: rm

The **rm** command (short for *remove*) deletes unwanted files. The system does not save copies of the files it removes. When you execute this command, your file is removed permanently.

If you want to delete a file from a directory belonging to another user, you must have write permission on the directory.

To use this command, type **rm** followed by one or more filenames. If you specify more than one file, leave a space between each filename. For example, to remove the *outline* and *table* files from your current directory, you would type this command:

```
$ rm outline table Return
```

You can use the **–i** option to make sure you don't remove a file accidentally. When you specify this option on the command line, the system prints the filename followed by a question mark, as shown on the following screen.

```
$ rm -i notices  Return
notices: ?
```

If the correct filename appears, answer **y** to delete the file. If the filename is incorrect, answer **n** to cancel the **rm** command.

The **–r** option deletes a directory and all files and subdirectories it contains. Be careful when using this command; it can delete many files in a single step. You might want to combine the **–r** option with the **–i** option to tell the system to prompt you before it removes a file or directory. For example, to delete the *seminar* directory and the files it contains, you could type the command shown on the following screen.

```
$ rm -ir seminar  Return
directory seminar: ? y  Return
seminar/schedule: ? y  Return
seminar/reservations: ? y  Return
seminar: ? y  Return
$
```

Other Information

rm(1)

3.2.15 Changing File and Directory Permissions: chmod

When you create a file or directory, the system assigns default permissions for the owner, the owner's group, and all other users. These default permissions are determined by the **umask** command, which the system administrator typically sets on a system-wide basis.

If necessary, you can change the permissions assigned to a particular file or directory. For example, you might want to keep certain files private and not allow anyone else to read or write to them, or you might want to allow members of your group to make changes to one of your files. You could also share a program with all users on your system by giving them permission to execute the file containing the program.

The **chmod** command (short for *change mode*) changes file or directory permissions. You can use this command only with your own files or directories. When you invoke **chmod**, you can use either symbols or octal notation to change permission levels.

Changing Permissions with Symbols

The **chmod** command recognizes symbols for read, write, and execute permission, and for the owner, group, and other users. These symbols represent permission levels:

r	Read permission
w	Write permission
x	Execute permission

These symbols represent the owner of the file or directory, the owner's group, and all other users:

u	You, the owner
g	Members of your group
o	Other users
a	All users (owner, group, and other users)

To add a permission, type a command in the following form. Be sure not to leave a space between *who* (the symbol for the owner, group, or other users) and the permission you are assigning.

$ **chmod** *who+permission file(s)*

For example, to give your group write permission on your *schedule* file, you would type this command:

$ **chmod g+w schedule** Return

As another example, you might own a directory named *movie.reviews* that contains several files describing current movies. To allow all users to add new files to this directory, you must give them write permission on the directory.

$ **chmod a+w movie.reviews** Return

If you want to remove a permission for yourself, your group, or all other users, type a command in the form shown below. Be sure not to leave a space between *who* (the symbol for the owner, group, or other users) and the permission.

$ **chmod** *who–permission file(s)*

For example, you might have a file named *display* containing a program you have written. To make sure you don't change the program accidentally, you can remove write permission for yourself, as shown below.

$ **chmod u–w display** Return

You can add or remove permissions in a single command for any combination of yourself, your group, and other users. For example, to remove write permission from your *schedule* file for your group and all other users, you would type this command:

$ **chmod go–w schedule** Return

After using **chmod** to change the permission level for a file or directory, you can execute **ls –l** to verify that the new permission level is correct.

Changing Permissions with Octal Notation

In octal notation, permissions for a particular file or directory are expressed as a three-digit number such as 664. The first digit specifies permissions for the owner, the second digit specifies permissions for the group, and the third digit specifies permissions for all others. Each digit is a sum based on the following values:

4 = Read permission
2 = Write permission
1 = Execute permission
0 = No permission

To determine the value to assign for the owner, group, or other users, simply add together the numbers representing the appropriate permission levels. For example, if the owner should have read and write permission, you would add 4 (read permission) and 2 (write permission). Then assign the value 6 to the digit representing the owner. Similarly, if other users should have execute permission only, you would use 1 as the digit representing other users.

The following list shows the possible values for each digit.

0 No permission
1 Execute only
2 Write only
3 Write and execute (1+2)
4 Read only
5 Read and execute (4+1)
6 Read and write (4+2)
7 Read, write, and execute (4+2+1)

To use octal notation, type **chmod**, then type the three-digit number specifying permissions for the owner, group, and other users. Then type the file or directory name. For example, to give yourself read and write permission on your *memories* file and to give the group and all other users read permission only, you would type this command:

$ **chmod 644 memories** Return

If you wanted to give yourself read, write, and execute permission on your *starship* directory and give your group and other users execute permission only, you would type this command:

$ **chmod 711starship** Return

Similarly, if you (the owner) and your group should have read and write permission on the *letters* file and other users should have no permissions, you would type this command:

$ **chmod 660 letters** Return

Other Information

chmod(1), **umask**(1)

3.2.16 Counting Lines, Words, and Characters in a File: wc

The **wc** command (short for *word count*) reports the number of lines, words, and characters in the files named on the command line. If you specify more than one file, **wc** counts the number of lines, words, and characters in each file and then totals the counts. You can also direct **wc** to count only lines, words, or characters.

To determine the number of lines, words, and characters in a file, type **wc** followed by the filename. For example, to count the lines, words, and characters in the *johnson* file in the current directory, type the command shown on the following screen.

```
$ wc johnson  Return
24      66      406 johnson
$
```

The system responds with three numbers followed by the name of the file. The first number means that the file contains 24 lines, the second number means that the file contains 66 words, and the last number means that the file contains 406 characters.

If you want to count the lines, words, and characters in more than one file, type the names of the files, separated by spaces, on the command line. For example, we typed **wc** on the following screen to count the lines, words, and characters in the *johnson* and *sanders* files in the current directory.

```
$ wc johnson sanders  Return
24      66      406 johnson
28      92      559 sanders
52     158      965 total
$
```

The first line reports the number of lines, word, and characters in the *johnson* file; the second line reports the lines, words, and characters in the *sanders* file. The last line shows the totals for both files.

If you want to count only lines, words, or characters, use one of the following options on the command line.

wc –l Counts lines only

wc –w Counts words only

wc –c Counts characters only

For example, the following command uses the **–l** option to report the number of lines in *sanders*.

```
$ wc -l sanders Return
28 sanders
```

If we specified the **–w** option, the command would report the number of words in the file; the **–c** option would report the number of characters.

Other Information

wc(1)

3.2.17 Identifying Differences Between Files: diff

The **diff** command reports all differences between two files. If the files are identical, the shell prompt appears. If the files are different, **diff** tells you how to use **ed** (line editor) commands to change the first file to match the second.

To use the command, type **diff** followed by the names of the files. For example, the following command compares the *johnson* and *mcdonough* files. The *mcdonough* file contains the same letter that is in the *johnson* file, with appropriate changes for a different recipient.

```
$ diff johnson mcdonough Return
```

The **diff** command identifies the lines in the first file that must change to match the second file by flagging them with a less-than (<) symbol. A greater-than (>) symbol flags the corresponding lines in the second file, as shown on the following screen.

```
3,6c3,6
< Mr. Ron Johnson
< Layton Printing
< 52 Hudson Street
< New York, N.Y.
---
> Mr. J.J. McDonough
> Ubu Press
> 37 Chico Place
> Springfield, N.J.
9c9
< Dear Mr. Johnson:
---
> Dear Mr. McDonough:
```

The first line of output from **diff** is as follows:

```
3,6c3,6
```

This line means that if you want *johnson* to match *mcdonough*, you must change (c is the **ed** change command) lines 3 through 6 in *johnson* to lines 3 through 6 in *mcdonough*. The **diff** command then prints lines 3 through 6 from both files.

Next, **diff** tells you to change line 9 (`9c9`) and shows this line from both files.

If you make the changes **diff** indicates, the *johnson* file will be identical to the *mcdonough* file. Remember, the **diff** command identifies differences between files. If you want an identical copy of a file, use the **cp** command.

Although **diff** specifies **ed** commands, you can use any text editor to make any necessary changes.

The **diff3** command is similar to **diff**; however, it compares the contents of three files.

Other Information

Chapter 8, "The **ed** Line Editor"

diff(1), **diff3**(1)

3.2.18 Searching a File for a Pattern: grep

The **grep** command searches through a file for a specific word, phrase, or group of characters. **grep** is short for *globally search for a regular expression and print*. Put simply, a regular expression is any pattern of characters (for example, a word, a phrase, or an equation) that you specify.

To use this command, type **grep** followed by the search pattern and the name of the file. For example, to locate lines in the *johnson* file containing the word `automation`, you would type the following command. The system then prints all lines containing the search pattern.

```
$ grep automation johnson Return
and office automation software.
```

If the search pattern contains multiple words separated by spaces or a character that conveys special meaning to the operating system (such as $, | , *, ?, ^, (,), \, [,]), you must enclose the entire pattern in single quotes. For example, you might want to locate the lines containing the pattern `office automation`. The command line and the system's response are as follows:

```
$ grep 'office automation' johnson Return
and office automation software.
```

If you want **grep** to search more than one file, type the names of the files on the command line, leaving a space between each name, as shown on the following screen.

```
$ grep 'office automation' johnson sanders Return
johnson:and office automation software.
sanders:We were talking about office automation.
sanders:Their office automation project is on schedule.
```

The output includes all lines from the *johnson* and *sanders* files containing the pattern `office automation`.

There are several useful options for the **grep** command. The **-n** option prints a line number with each line of output.

```
$ grep -n 'office automation' johnson Return
15    and office automation software.
```

The **–i** option causes **grep** to ignore the distinction between uppercase and lowercase letters.

```
$ grep -i 'office automation' johnson Return
and office automation software.
Office automation will be complete by September.
```

The **–c** option prints the number of lines containing the pattern.

```
$ grep -c 'office automation' johnson Return
1
```

The **–v** option prints all lines that do not contain the pattern. For example, if you wanted to print the lines in the *johnson* file that do not contain the pattern `office automation`, you would enter this command:

```
$ grep -v 'office automation' johnson Return
```

There are also variations of the **grep** command called **egrep** and **fgrep** that you might find useful. The **fgrep** command interprets special characters literally; **egrep** recognizes a larger set of special characters than **grep** does. Both of these variations are described in the *Reference Manual.*

Creating a Regular Expression

The search pattern for the **grep** command (and also for **pg**) is called a regular expression. This pattern can be a simple character string, such as `automation`, that matches only itself. The pattern can also include special characters that cause the pattern to match many character strings. When a regular expression includes a special character, it matches the longest possible string starting as close to the beginning of the line as possible.

You can use the following special characters in a regular expression.

Period. A period matches any character.

Example	*Meaning*	*Matches*
te.m	Matches strings beginning with `te`, followed by any character, followed by `m`	team, teem
.vy	Matches strings with any number of characters preceding `vy`	ivy, levy

Asterisk. An asterisk matches zero or more occurrences of the character that precedes it. If a period precedes the asterisk, the asterisk matches all preceding characters.

Example	*Meaning*	*Matches*
do*g	Matches `d` followed by zero or more `o`'s followed by `g`	dg, dog, doog, dooog
s.*ing	Matches `s` followed by zero or more characters followed by `ing`	sing, sting, string, started talking

Caret. When an expression has a caret (^) as its first character, it matches lines starting with that expression.

Example	*Meaning*	*Matches*
^St	Matches lines beginning with `St`	Start the band...

Dollar Sign. When an expression has a dollar sign ($) as its last character, it matches lines ending with that expression.

Example	*Meaning*	*Matches*
!$	Matches lines ending with !	That's all!

Brackets. Brackets define a set of characters, such as [abc]. The expression matches any one of these characters. If the first character in the brackets is a caret (^), the expression matches any character not in the brackets.

Example	*Meaning*	*Matches*
[tT]om	Matches `t` or `T` followed by om	Tom, tomato
'book [2-5]'	Matches `book` followed by a space followed by any of the numbers 2 through 5	book 2, book 4, book 5
[^A-Z]	Matches any character that is not an uppercase letter	1,6, %, a, b, c

Removing the Special Meaning from a Character. If you want any of these special characters to be interpreted literally in a search pattern, you can remove the special meaning by preceding the character with a backslash. For example, if you want to search for the expression `*struck`, you must type the expression as ***struck**. The backslash removes the special meaning from the asterisk, allowing the command to interpret the character literally.

Other Information

egrep(1), **fgrep**(1), **grep**(1)

3.2.19 Locating Files: find

The **find** command locates files that match the criteria you specify. When you use this command, you must provide a list of directories to search. The command takes this form:

find *pathname-list expression*

The *pathname-list* is a list of the directories to be searched. The **find** command will also search all subdirectories of the directories named in the list. The *expression* contains the criteria a file must meet to be selected by the **find** command. This command does not generate any output unless you include criteria telling it to do so.

Following is a partial list of the criteria you can use in an expression. Each criterion begins with a hyphen (–). The character *n* represents a decimal integer.

–name *pattern*
A file satisfies this criterion if its filename matches *pattern*. You can use special characters such as asterisk (*) or question mark (?) in the pattern, but you must surround the pattern in quotes, such as **'gold*'**.

–type *filetype*
A file satisfies this criterion if it is the type of file specified by *filetype*, which can be any of the following letters:

b	Block special file
c	Character special file
d	Directory
p	Plain file
f	FIFO (named pipe)
l	Symbolic link

–links *n*
A file satisfies this criterion in it has the number of links specified by *n*.

–user *name*
A file satisfies this criterion if it is owned by user *name*.

–group *name*
A file satisfies this criterion if it belongs to the group specified by *name*.

–atime *n*
A file satisfies this criterion if it has been accessed within *n* days.

–mtime *n*
A file satisfies this criterion if it has been modified within *n* days.

–print
All files meet this criterion. This criterion causes **find** to print the pathname of the file. If this is the only criterion in the expression, **find** will print the names of all files in the specified directory. If other criteria are included in the expression, **find** will print a filename only if the file meets the criteria preceding **–print**.

–exec *command* **\;**
A file satisfies this criterion if the *command* returns a zero (true value) as its exit status. You must follow *command* with a backslash and a semicolon (\;). You can include a pair of braces ({}) in the command. **find** then substitutes the name of the file being tested for the braces. (See the example at the end of this section.)

Typically this criterion is placed at the end of an expression. It will then execute the command for those files meeting the previous criteria.

–ok *command* **\;**
This criterion is the same as **–exec**, however it will display the command to be executed, followed by a question mark. It executes the command only if you answer **y**.

If you want to include multiple criteria in an expression, keep these rules in mind:

- If a file must meet both criteria, separate the criteria by a space.
- If a file can meet either or both criteria, separate the criteria with **–o**.
- If you want to negate a criterion, precede it with an exclamation point.
- The **find** command evaluates an expression from left to right unless you use parentheses to group criteria.
- If you include a special character (such as *, ?, (,), [,]) within an expression and do not want the character to have its special meaning, you must precede the character with a backslash, such as \(.

- Use spaces to separate arguments. There must be a space on each side of a parenthesis, exclamation point, criterion, or other element. If you used a backslash to remove the special meaning from a character, insert spaces on each side of the pair of characters.

The following examples use the **find** command. The first example uses **find** to locate all files beginning with *memo*. The **–print** criterion tells **find** to print the names of those files meeting the **–name** criterion. The dot (.) tells **find** to search the current directory and its subdirectories.

```
$ find . -name 'memo*' -print Return
```

In the next example, **find** searches the */user1* directory and its subdirectories for files belonging to either user *starship* or user *jmrs* that have not been accessed in the last 20 days (**–atime +20**). It then asks if you want to remove each file (**–ok rm {}**) that meets this criteria.

```
$ find /user1 \( -user starship -o -user jmrs \) -atime +20 -ok rm {} \;
< rm ...   /user1/starship/letters/sanders >? y
< rm ...   /user1/jmrs/meetings/boston >? n
$
```

Other Information

find(1)

3.2.20 Sorting and Merging Files: sort

The **sort** command can sort the lines in a file and can merge files. You can sort a single file, sort multiple files and then merge them, or merge previously sorted files.

The **sort** command sorts the lines in a file in the order you specify. If you invoke **sort** with no options, it sorts the lines in this order:

- Lines beginning with numbers are sorted by digit and listed before lines beginning with letters.
- Lines beginning with uppercase letters are listed before lines beginning with lowercase letters.
- Lines beginning with symbols such as *, %, or @ are sorted by the symbol's ASCII representation.

For example, you might have two files, *group1* and *group2*, each containing a list of names, as shown on the following screen.

```
$ cat group1  Return
Smith, Allyn
Jones, Barbara
Cook, Kenny
Moore, Pamela
Wolf, Randy
$ cat group2  Return
Frank, M. Jay
Nelson, James
West, Donna
Hill, Charles
Morgan, Kristine
$
```

You can use the **sort** command to sort each list alphabetically and then merge the two lists into one. Simply type **sort** followed by the names of the files.

```
$ sort group1 group2  Return
```

The **sort** command prints its output on your screen, as shown here:

```
$ sort group1 group2  Return
Cook, Kenny
Frank, M. Jay
Hill, Charles
Jones, Barbara
Moore, Pamela
Morgan, Kristine
Nelson, James
Smith, Allyn
West, Donna
Wolf, Randy
$
```

If the files are already sorted, you can use the **-m** option to merge the files. For example, if you had previously sorted the *group1* and *group2* files, you could use this command to merge the files:

```
$ sort -m group1 group2  Return
```

The **-c** option determines if the file specified on the command line is sorted according to the ordering rules. This option does not produce any output. However, it will display a message if the file is not sorted. For example, if you used this option before *group1* was sorted, **sort** would display the following message:

```
$ sort -c group1  Return
sort: disorder: Jones, Barbara
```

Changing the Sort Order

The following options to the **sort** command change the order in which the lines of a file are sorted.

–d (Dictionary order)
sort uses only letters, digits, and blanks (either spaces or tabs) in comparisons. Punctuation and Ctrl characters are ignored.

–f (Ignore case)
sort considers lowercase letters to be uppercase letters.

–i (Ignore non-printing characters)
sort does not use non-printing characters in comparisons.

–M (Sort as months)
sort shifts the first three non-blank characters in the field to uppercase, then sorts the file in the order *JAN*, ... *DEC*. Invalid entries print first in the sorted list.

–n (Sort in numeric order)
sort recognizes minus signs and decimal points and sorts in arithmetic order.

–r (Reverse the sort order)
sort reverses the sort order so that *z* precedes *a*.

Sending Sorted Output to a File

You can use the **–o** command line option to send the sorted output to a file. Type **sort –o**, then type the name of the file you want to contain the output. The following command sends the output to the *grp.sorted* file.

```
$ sort –o grp.sorted group1 group2 Return
```

You can also use the **–o** option to direct the output back to one of the files being sorted. For example, if you want to sort the *group3* file and place the output in the *group3* file, you would type the following command. The sorted output overwrites the original contents of *group3*.

```
$ sort –o group3 group3 Return
```

Sorting on Fields

In the examples shown so far, the **sort** command sorts the lines in a file according to their first characters. You can also use the contents of a specific part of the line as the basis for sorting lines. The parts of a line are called *fields*. Each field either contains a character string or is empty. Fields can be separated by spaces or tabs or by another character such as a colon (:). The character used to separate fields is called a *field delimiter* or *field separator*. For example, the following *group* file contains six fields separated by spaces.

```
$ cat group Return
Allen Smith Portland 97219 swing 899-8434
Barbara Jones Tigard 97223 swing 408-5421
Kenny Cook Portland 97225 days 888-4560
Pamela Moore Aloha 97006 days 928-0705
Randy Wolf Portland 97219 days 899-3459
```

We'll sort the lines in this file according to the zip code, which is in the fourth field.

The notation *+pos –pos* indicates the field to be sorted. The *+pos* parameter tells **sort** the number of fields to skip. In this example, the zip code is in the fourth field. To tell **sort** to skip the first three fields and sort the fourth field, you would type **+3**.

The *–pos* parameter tells **sort** where to stop the sort. In our example, we want to sort only the fourth field. To tell **sort** to stop sorting at the end of the fourth field, you would type **–4**. (If you omit the *–pos* parameter, sorting will go to the end of the line.)

Following is the complete command to sort the *group* file so the lines are in order according to zip code.

```
$ sort +3 –4 group Return
```

The **+3** tells **sort** to skip the first three fields; the **–4** tells **sort** to stop sorting at the end of the fourth field. The output from the sort looks like this:

```
$ sort +3 -4 group Return
Pamela Moore Aloha 97006 days 928-0705
Allen Smith Portland 97219 swing 899-8434
Randy Wolf Portland 97219 days 899-3459
Barbara Jones Tigard 97223 swing 408-5421
Kenny Cook Portland 97225 days 888-4560
```

You can specify multiple sort fields. The **sort** command first sorts the file by the first field you specify. If several lines have the same value for that field, **sort** then sorts those lines according to the next field you specify. To sort the *group* file from the previous example first by zip code and then by phone number, you would type the command shown on the following screen.

```
$ sort +3 -4 +5 group Return
Pamela Moore Aloha 97006 days 928-0705
Randy Wolf Portland 97219 days 899-3459
Allen Smith Portland 97219 swing 899-8434
Barbara Jones Tigard 97223 swing 408-5421
Kenny Cook Portland 97225 days 888-4560
```

The **+3 –4 +5** notation tells **sort** to first sort the fourth field (zip code), then sort any lines having the same zip code according to the sixth field (phone number). Because the lines for Randy Wolf and Allen Smith have the same zip code, they are also sorted according to phone number.

Options for Specifying a Field Delimiter. By default, blanks (either spaces or tabs) separate the fields on a line. If a line includes a series of blanks, the first blank is considered to be the field separator; the remaining blanks are considered to be the first characters in the next field. You can use the following command line options to override these default values:

–b This option tells **sort** to ignore leading blanks when it is determining where a field begins.

–t*x* This option specifies a field separator other than a blank. For example, if the fields are separated by colons, you would type **–t:** on the command line. (The example in the next section uses this option.)

Sorting on a Specific Character within a Field. You can begin or end a sort on a particular character within a field. To indicate this, use the notation *+m.n* or *–m.n* where *m* represents the number of fields to skip and *n* represents the number of characters to skip. For example, to begin a sort on the fourth character of the third field, you would type **+2.3**. This tells **sort** to skip two fields and the first three characters of the third field.

A *–m.n* notation tells **sort** to end the sort *n* number of characters after the field indicated by *m*. For example, to end the sort three characters after the end of the fourth field, you would type **–4.3**. The **sort** command counts the field separator as a character.

To show how this works, we'll sort the following file according to last name.

```
$ cat group2 Return
A. Smith:Portland:97219:swing:899-8434
B. Jones:Tigard:97223:swing:408-5421
K. Cook:Portland:97225:days:888-4560
P. Moore:Aloha:97006:days:928-0705
R. Wolf:Portland:97219:days:899-3459
```

To sort this file according to last name, **sort** needs to skip the first three characters of the first field. To indicate this, use the notation **+0.3**, as shown on the following screen. Because colons separate the fields in this file, we also used the **–t** option.

```
$ sort -t: +0.3 group2 Return
K. Cook:Portland:97225:days:888-4560
B. Jones:Tigard:97223:swing:408-5421
P. Moore:Aloha:97006:days:928-0705
A. Smith:Portland:97219:swing:899-8434
R. Wolf:Portland:97219:days:899-3459
```

Other Information

sort(1)

3.2.21 Cutting Fields or Characters from a File: cut

The **cut** command extracts part of each line in a file. It can extract either specific fields or characters. To specify characters, use the **–c** option followed by the first and last position the characters occupy, counting from the left-hand margin.

For example, you might want to see only the dates from a file called *birthdays*. The file contains the following list:

```
Anne....12/26
Klaus...7/4
Mary....10/18
Peter...11/9
Nandy...4/23
Sam.....8/12
```

The birthdays appear between the 9th and 13th spaces on each line. To print the dates, type the command shown on the following screen.

```
cut -c9-13 birthdays  Return
12/26
7/4
10/18
11/9
4/23
8/12
```

To print specific fields from a file, use the **–f** option followed by a list of fields. For example, to print fields 4 and 5, you would type **–f4,5**. The **cut** command assumes that fields are separated by tabs. If your file uses a different character to separate fields, include the **–d** option followed by that character. For example, if colons separate the fields, you would include **–d:** on the command line. If the field separator is a space or a character having special meaning to the shell, enclose the separator in quotes, such as **–d' '**.

As an example, you might have a file containing the following information:

```
$ cat group2 Return
A. Smith:Portland:97219:swing:899-8434
B. Jones:Tigard:97223:swing:408-5421
K. Cook:Portland:97225:days:888-4560
P. Moore:Aloha:97006:days:928-0705
R. Wolf:Portland:97219:days:899-3459
```

To print the names and phone numbers from this file, you need to cut fields 1 and 5, as shown on the following screen. Because colons separate the fields, we included the **–d** option on the command line.

```
$ cut -d: -f1,5 group2 Return
A. Smith:899-8434
B. Jones:408-5421
K. Cook:888-4560
P. Moore:928-0705
R. Wolf:899-3459
```

You must specify either the **–c** or **–f** option when you use the **cut** command. If you want to save the output from **cut**, redirect it to another file, as shown below.

```
$ cut -d: -f1,5 group2 > grp2.phone Return
```

Other Information

cut(1)

3.3 Changing to Another Group: newgrp

When your login account was set up, you were assigned to a specific group of users. When you log in, you are placed in that group.

When several users need access to each other's files, the system administrator can create a special group for those users. The **newgrp** command allows you to switch from your regular group to that group. To use this command, type **newgrp** followed by the name of the group you want to move to. For example, to switch from your regular group to the products group, you would type this command:

```
$ newgrp products (Return)
```

To return to your original group, type this command:

```
$ newgrp (Return)
```

While you are in the new group, you have only the permissions that apply to that group. You will remain in the new group until you either use the **newgrp** command again or log out of the system.

To see a list of the groups you belong to, type **groups** and press (Return).

Other Information

groups(1), **newgrp**(1)

3.4 Disk Quotas

Many sites set up disk quotas for each user. There are two types of quotas: a file quota and a disk usage quota. A file quota determines the maximum number of files you can create on a specified filesystem. A disk usage quota determines the maximum amount of disk storage space those files can occupy. (Disk storage space is measured in 1K blocks; each block contains 1024 bytes.)

You can be assigned a file quota only, a disk usage quota only, or both types of quotas. You can also be assigned different quotas on different filesystems.

3.4.1 Limits

There are two limits for each quota: a hard limit and a soft limit. Your disk usage or number of files must remain below the soft limit, which typically is lower than the hard limit.

When you reach a soft limit, you will receive a warning and a timer will be set. You must reduce your disk usage (or number of files) to below the soft limit within the allotted time. If you don't do this before the time expires, the system will treat the soft limit as a hard limit until your disk usage (or number of files) falls below the soft limit.

When you reach a hard limit, you will receive a message on your terminal. (The message is not repeated.) If you reached the limit for your file quota, you will be unable to create more files. If you reached the limit for your disk usage quota, you will be unable to use more blocks. To correct this situation, you must reduce the number of files or blocks you hold to below the soft limit.

For example, if you are creating a file and reach the file limit, an error message appears and the file is not created. At this point, you must remove files until you fall below the soft limit. You can still add blocks to existing files if the number of blocks used hasn't reached the disk usage limit.

3.4.2 Checking Quotas and Limits

When you log in, your quotas and limits are checked automatically. You will receive a message if you are over a limit. You can also check your own quotas and limits at any time. To do this, issue the **quota** command as shown below. You cannot check anyone else's limits.

```
$ quota Return
```

If you are under your limits, you will not receive any output from this command. If you are over a soft limit, you will be told how much you are over the limit and the time left to correct the situation. If you are over a hard limit, you will be told the number of files or disk blocks that must be removed to bring you under the soft limit.

The **-v** option to the **quota** command provides more information. Following is a sample of its output for a user having both a disk usage quota and a file quota.

```
$ quota -v Return
Disk quotas for ramona (uid 152):
Filesystem usage quota limit timeleft  files quota limit timeleft
/user1       128   200   300              15    30    40
/user2        63   200   300               7    30    40
```

For each filesystem, the output shows the user's current disk usage (the `usage` column) and number of files (the `files` column). The `quota` column contains the user's soft limit; the `limit` column contains the user's hard limit. If the user is over a soft limit, the `timeleft` column will specify the amount of time remaining to correct the situation. If the time period is up or a hard limit is exceeded, the message `EXPIRED` will appear in the `timeleft` column.

3.4.3 Responding After Reaching a Hard Limit

If you reach a hard limit, typically you must abort the process or processes in progress on the filesystem where you reached the limit. Then remove enough files to bring your disk usage below the quota and retry the failed program.

The case is different when you are working with an interactive program such as **ed** or **vi** and a write fails because it exceeds your quota. The write attempt that generated the "quota exceeded" message probably also truncated the file in the editor. If you abort the editor in these circumstances, you will lose not only your recent changes, but possibly some or all of the file itself. You can use any of these methods to exit from this situation safely:

- Use an editor shell escape command. Typing an exclamation mark (**!**) starts up your shell program and allows you to issue a single shell command. You will return to the editor when the command completes. You might need to execute several shell escape commands to check your disk usage and remove surplus files. (For more information about the editor shell escapes, see Chapter 8, "The **ed** Line Editor," or Chapter 9, "The **vi** Screen Editor.")
- If you are using either the Korn shell or the C shell, you can press Ctrl-Z to suspend the editing session. The shell prompt then appears on your screen. You can issue as many commands as necessary to remove files and get below your quota limit. When you are ready to continue the editing session, type **fg** (for *foreground*) at the shell prompt to restart the editor. (For more information, see Chapter 7, "Managing Processes.")
- Write the file you are editing to another filesystem (perhaps */tmp*) where your quota has not been exceeded. After rectifying the quota problem, you can move the file back to its original filesystem.

3.4.4 Quotas and ptx/NFS

If your site is using ptx/NFS, you might be assigned quotas on a remotely mounted filesystem. These quotas work much the same as quotas on locally mounted filesystems. The important difference is that no warnings are printed when you reach a soft limit. The operating system does not print an error message when you reach a hard limit, and the program you are running may or may not give a message. This means that you cannot rely on error messages to remain within your limits.

Also, on remotely mounted filesystems, your system administrator can turn off the automatic quota check that takes place when you log in. If you want to check your quotas yourself, use the **quota –v** command.

Other Information

quota(1)

Chapter 4
Using the Shell

4.1 Introduction 4-1
Special Characters 4-1
Your Environment 4-1
The Bourne, Korn, and C Shells 4-1

4.2 Filename Substitution 4-2
Matching One Character 4-3
Matching All Characters 4-4
Matching One Character of a Set 4-5
Matching Characters Not In a Set 4-6

4.3 Running Commands in the Background 4-6

4.4 Executing Commands Sequentially 4-7

4.5 Input and Output Redirection 4-8
Redirecting Input 4-8
Redirecting Output to a File 4-9
Combining Background Mode and Output Redirection 4-9
Appending Output to an Existing File 4-10
Redirecting Output to Another Command 4-11
Testing a Pipeline: **tee** 4-12

4.6 Printing Arguments: echo 4-13

4.7 Using Command Output as an Argument 4-14

4.8 Removing Meanings from Special Characters 4-14
Using the Backslash 4-15
Using Single Quotes 4-16
Using Double Quotes 4-17
Interpreting Backquotes 4-17
Interpreting the Dollar Sign 4-18
Interpreting the Backslash 4-19

4.9 The Bourne and Korn Shell Environment 4-20
Environment Variables 4-20
Modifying Environment Variables 4-23
Setting a Variable for the Current Login Session 4-23

Setting a Variable in Your ***.profile*** 4-24
Executing ***.profile*** 4-25
Adding Commands to Your ***.profile*** 4-26
Setting Terminal Options: **stty** 4-26

4.10 Using the Korn Shell 4-28
Invoking the Korn Shell 4-28
The Korn Shell Environment 4-29
Editing the Command Line 4-30
Command History 4-31
Displaying Command History 4-31
Reexecuting Commands 4-32
Editing Command Lines 4-32
Searching for a Command 4-33
Creating Aliases 4-33
The **cd** Command 4-34
Using a Tilde to Represent a Home Directory 4-35
Command Substitution 4-36

4.11 Using the C Shell 4-37
Invoking the C Shell 4-37
The C Shell Environment 4-38
Environment Variables 4-38
C Shell Variables 4-39
Displaying the Value of a Variable 4-40
Setting Variables 4-41
Setting Environment Variables 4-41
Setting C Shell Variables 4-42
Adding Variables to Your ***.login*** or ***.cshrc*** File 4-42
Setting the TERM Variable 4-43
Changing Your Command Prompt 4-44
Executing ***.login*** and ***.cshrc*** 4-44
Adding Commands to Your ***.login*** File 4-45
Setting Terminal Options: **stty** 4-45
Creating Aliases 4-46
Command History 4-47
Reexecuting Commands 4-47
Searching for a Command 4-48
Modifying Your Last Command 4-48
Substituting Text in a Command 4-49
Using a Tilde (~) to Represent a Home Directory 4-49
Changing Directories 4-50

Chapter 4
Using the Shell

4.1 Introduction

This chapter shows how to use special characters on the command line to tell the shell to perform tasks such as input and output redirection. It also describes your shell environment and the shell programs available to you.

4.1.1 Special Characters

You can use special characters to tell the shell to perform specific tasks. For example, the greater-than sign (>) tells the shell to redirect the output from a command to a file. You can also use special characters such as the asterisk (*) and question mark (?) to match or abbreviate filenames. When the shell reads special characters on the command line, it does not interpret them literally, as text. Instead, it carries out the special meaning assigned to the characters. If you want to use a special character as text, you can remove the special meaning from the character.

4.1.2 Your Environment

When you log into the system, the shell sets up an *environment* for you. This environment consists of a set of variables that define values such as the type of terminal you are using and the command prompt that appears on your screen. You can use the default environment set up for you, or you can tailor it to meet your individual needs.

4.1.3 The Bourne, Korn, and C Shells

There are three shell programs available to you: the Bourne shell, the Korn shell, and the C shell. You can use any of these programs to interpret your commands, however the Korn and C shells also allow you to perform tasks such as reexecuting commands.

When your login account was set up, one of these shell programs was assigned as your default shell. When you log in, this program starts up and then processes each command you enter. The Bourne and Korn shells use a dollar sign (`$`) as the command prompt; the C shell uses a percent sign (`%`). If you are unsure of which program is your default shell, type this command:

```
$ echo $SHELL Return
```

(The sections of this chapter that describe your environment will explain this command.) If the Bourne shell is the default, `/bin/sh` will appear; if the Korn shell is the default, `/bin/ksh` will appear; and if the C shell is the default, `/bin/csh` will appear. Unless stated otherwise, the examples in this manual assume you are using the Bourne shell. However, most examples are also valid for the Korn and C shells.

4.2 Filename Substitution

When you type a command, you can use special characters (called *metacharacters* or *wild cards*) to represent part or all of a filename. For example, you might want to refer to all filenames beginning with a specific letter or to all filenames in your current directory.

Metacharacters can match a single character within a filename, or they can match a string of characters. When the shell reads a command line, it first substitutes the appropriate filenames in place of the metacharacters. This step is called *filename substitution* or *filename expansion*. The shell then executes the command.

The metacharacters are as follows:

?	Matches a single character
*	Matches a string of characters
[ ]	Matches any character within the brackets
[!]	Matches any character not within the brackets (works with Bourne and Korn shells only)

4.2.1 Matching One Character

The question mark (?) matches a single character in a filename. On the following screen, we used the command **ls chapter?** to list all files beginning with the string `chapter` and ending with any single character.

```
$ ls chapter?  Return
chapter1
chapter5
chapter9
$
```

You can use the question mark (?) metacharacter more than once in a filename. The following command lists filenames beginning with *chapter* and ending with any two characters.

```
$ ls chapter??  Return
chapter10
chapter11
chapter12
$
```

The question mark (?) can also occupy any position in the filename. The following command moves all files starting with *quarter*, followed by any character, and ending in *_87* to the *oldfiles* directory.

```
$ mv quarter?_87 /user1/tom/oldfiles  Return
```

4.2.2 Matching All Characters

The asterisk (*) matches any string of characters, including a null (empty) string. You can type an asterisk by itself, or you can include it in a filename.

If you type an asterisk only, it refers to all file and directory names in the current directory. For example, the following command moves all files from the current directory to user *sarah's* home directory.

```
$ mv * /user1/sarah Return
```

When you include an asterisk in a filename, the asterisk matches any characters in that position or following that position. It also matches no characters. For example, you might have several files named *report*, *report1*, *report1a*, *report1b.01*, and *report316*. If you type **report*** on the command line, you can refer to all filenames beginning with *report*. The asterisk matches any characters after the string `report`, including no characters at all.

```
$ ls report*  Return
report
report1
report1a
report1b.01
report316
$
```

The asterisk can match characters in any part of a filename. For example, the following command prints the names of all files in your current directory containing a lowercase `a`.

```
$ ls *a* Return
```

As another example, the following command will print a list of filenames that begin with F and end with E. The asterisk matches any number of characters between F and E. The shell prints the filenames in this order: numbers, uppercase letters, lowercase letters.

```
$ ls F*E  Return
F123E
FATE
FE
Fig3.4E
$
```

NOTE

The asterisk is a powerful character. Be careful when using it! For example, if you type **rm ***, *you will delete all files in your current directory.*

4.2.3 Matching One Character of a Set

If you want the shell to match one of several possible characters that can appear in a specific position in a filename, enclose the characters in brackets ([ ]). The characters grouped within brackets are called a *character class* and must be entered in either alphabetical or numerical order.

For example, if you include **[abc]** as part of a filename, the shell will look for filenames that include the letter `a`, the letter `b`, or the letter `c` in the specified position. The command on the following screen prints all filenames beginning with the letters `a`, `b`, or `c` and ending with `_list`.

```
$ ls [abc]_list  Return
a_list
b_list
c_list
$
```

A character class can also include a range of numbers or letters. For example, if you specify **chapter[2-4]**, the shell will match files named *chapter2*, *chapter3*, and *chapter4*. If you specify **[A-Z]**, the shell will look only for uppercase letters; if you specify **[a-z]**, the shell will look for lowercase letters.

4.2.4 Matching Characters Not In a Set

You can also use brackets to locate a filename that does not match the character class. To do this, precede the character class with an exclamation point (!). For example, to locate filenames that do not begin with lowercase letters, you could type this command:

```
$ ls [!a-z] Return
```

This use of brackets works only in the Bourne and Korn shells.

4.3 Running Commands in the Background

Commands can execute in either *foreground* or *background* mode. These modes determine if you can use your terminal while the command is running. When a command executes in the foreground, you cannot enter another command until the current command completes. The commands shown so far in this manual ran in the foreground.

When a command runs in the background, the system displays the shell prompt immediately. You can then use your terminal for other tasks while the command executes. To place a command in the background, first type the command. Then type an ampersand (&), which tells the shell to execute the command in the background. The **grep** command on the following screen is running in the background.

```
$ grep delinquent accounts &  Return
21940
$
```

When you place a command in the background, the operating system prints a process identification (PID) number (`21940` in the example). You can use this number to check the status of the command or to stop its execution (see Chapter 7 for more information). The shell prompt (`$`) on the last line means you can type other commands while **grep** is running in the background.

Executing a command in the background affects only the availability of your terminal; it does not affect the output from the command. The command will still print its output on your terminal screen unless you redirect the output to a file, as described later in this chapter.

NOTE

Some commands require you to enter additional information after you have typed the command line and pressed Return. *For example, if you use* **cat** *to create a file, the command expects you to type the contents of the file and then press* Ctrl-D. *Do not run these commands in the background.*

4.4 Executing Commands Sequentially

The shell allows you to type multiple commands on a single command line. This procedure is called *sequential execution*. When you type several commands on the same command line, use semicolons (;) to separate the commands. The shell executes the commands in the order they appear in the command line and all output appears on the screen.

The following command line contains three commands separated by semicolons.

```
$ cd; pwd; ls Return
```

The shell executes the commands in this order:

1. **cd** changes your location to your login directory.
2. **pwd** prints the full pathname of your login directory.
3. **ls** lists the files in your login directory.

4.5 Input and Output Redirection

Many commands take their input from the terminal and most commands print their output on the terminal. However, the shell lets you reassign command input and output to other files and programs. This is known as *redirection*. The following special characters redirect input and output:

- The less-than sign (<) tells the shell to take the input for a command from a file.
- The greater-than sign (>) tells the shell to send command output to a file.
- Two greater-than signs (>>) tell the shell to append command output to an existing file.
- The pipe (|) tells the shell to use the output from one command as the input for another command.

4.5.1 Redirecting Input

If you want the input for a command to come from a file instead of your terminal, you can redirect the input on the command line. To do this, type the command followed by a less-than sign (<) and the name of the file that will supply the input.

command < *file*

For example, you might want to use the **mailx** command (described in Chapter 10) to send a message to user *ted*. If you already have the message in a file named *report*, you can avoid retyping the message by specifying the file as the source of input.

$ **mailx ted < report** Return

You can also redirect input to come from your terminal rather than a file. If you issue a command that takes its input from a file and you do not provide a filename, the command will expect its input from the terminal. For example, if you issue the **sort** command and don't supply a filename, you can type the items to be sorted. After typing the last item, press Ctrl-D to end the command. The **sort** command will then display the sorted list on your screen.

```
$ sort Return
potatoes Return
eggs Return
onions Return
mayonnaise Return
Ctrl-D
eggs
mayonnaise
onions
potatoes
```

4.5.2 Redirecting Output to a File

The shell allows you to redirect command output to a file. This feature is useful if you want to save the output or you don't want it to appear on your screen. To redirect output, type the command followed by a greater-than sign (>) and the name of the file to receive the output.

command > *file*

The following line uses redirection to save the output from **ls –l**:

$ **ls –l > ls.output** Return

Before redirecting output to a file, make sure that a file by that name does not already exist. If you redirect the output to an existing file, the shell will overwrite the contents of the existing file with the output from your command. The shell does not warn you that it will overwrite the original file.

Combining Background Mode and Output Redirection

When you run a command in the background, its output will still print on your screen. If you are using your terminal to perform other tasks, you will be interrupted when the output appears. However, if you redirect that output to a file, you can work undisturbed.

The following **grep** command runs in the background and its output is redirected to a file called *testfile*. Running the command in this manner allows you to use your terminal for other work and then examine the output in *testfile* later on.

$ **grep test schedule > testfile &** Return

4.5.3 Appending Output to an Existing File

You can also use output redirection to append command output to the end of an existing file. To perform this type of redirection, type the command followed by two greater-than signs (>>) and the name of the file to receive the output. If the file does not exist, it is created.

command >> *file*

The following example shows how to append output from the **cat** command to an existing file. We first executed the **cat** command on two files.

```
$ cat spring  Return
Spring is a lovely time of year.
The birds are nesting,
The sun draws near.
$
$ cat summer  Return
Summer is full of joy.
The grain ripens and time moves
Almost not at all.
$
```

We then executed **cat** on the *summer* file and redirected its output to *spring*.

```
$ cat summer >> spring  Return
```

This step appended the contents of the *summer* file to the end of the *spring* file.

```
$ cat spring  Return
Spring is a lovely time of year.
The birds are nesting,
The sun draws near.
Summer is full of joy.
The grain ripens and time moves
Almost not at all.
$
```

4.5.4 Redirecting Output to Another Command

The | character is called a *pipe*. Pipes allow you to use the output of one command as the input for another command. When you use the pipe character, the command line is called a *pipeline*.

Pipelines have this format:

command1 | *command2* | *command3...*

The shell uses the output from *command1* as the input for *command2*. It then uses the output from *command2* as the input for *command3*, and so on.

You can create a pipe between any two commands, provided the first command writes its output to the terminal and the second command reads its input from the terminal.

Pipelines can eliminate the need to redirect command output to temporary files. For example, you might want to paginate a file and send the paginated copy to user *david*. If you use a pipeline, you can issue a single command to perform this task.

$ **pr report | mailx david** Return

If you don't use a pipeline, this task requires three steps:

1. Issue the **pr** command and send its output to a temporary file:

   $ **pr report > report.tmp** Return

2. Mail the file to user *david*:

   $ **mailx david < report.tmp** Return

3. Remove the temporary file:

```
$ rm report.tmp Report
```

Pipelines also increase the versatility of individual commands. As an example, the **cut** command is typically executed on a file. However, you can use a pipeline to run this command on the output from other commands. This is useful if you want only part of the information generated by another command.

For example, you might want to print only the time. However, the **date** command also prints the date and the day of the week.

```
$ date Return
Sat Dec 17 13:12:32 EST 1988
```

The time appears in the 12th through the 19th spaces on the line. To print the time only, pipe the output from **date** into **cut**, using the **–c** option to specify spaces **12–19**. The command line and its output will look like this:

```
$ date | cut –c12-19 Return
13:14:56
```

Testing a Pipeline: tee

You can use the **tee** command to debug pipelines. It passes the input it receives to the next command and saves a copy of the input in the file you specify. This command is useful when you include a pipeline in a shell program.

The **tee** command has this format:

command1 | **tee** *file* | *command2*

The **tee** command receives its input from *command1*. It does not change the input, it simply passes it on to *command2*. However, it saves a copy in *file*.

As an example, you might want to check the output from the **grep** command in the following pipeline:

```
who | grep steve | cut –c1-9 Return
```

Without disturbing the rest of the pipeline, you can use **tee** to copy the output from **grep** into a file called *check*. You can then issue the **cat** command to look at the contents of the *check* file.

```
$ who | grep steve | tee check | cut -c1-9 Return
steve
$ cat check Return
steve    tty61    Apr 10    11:30
$
```

Other Information

tee(1)

4.6 Printing Arguments: echo

To understand the examples in the following sections, you should be familiar with the **echo** command. This command simply prints its arguments on the screen. It is commonly used in shell programs.

```
$ echo one two three  Return
one two three
$
$ echo baseball  Return
baseball
```

Other Information

echo(1)

4.7 Using Command Output as an Argument

You can use the output from any command as an argument to another command. This procedure is called *command substitution*. To use this feature, place backquotes (`...`) around the command whose output will be used as an argument.

In the following example, we used the output from the **date** command as an argument to the **echo** command.

```
$ echo Today is `date`  Return
Today is Sat Dec 17 13:12:32 EST 1988
$
```

You can include several commands within backquotes. For example, the output from the **date** and **cut** pipeline command is the argument for the following **echo** command. This command prints the current time.

```
$ echo `date | cut -c12-19`  Return
11:30:53
```

4.8 Removing Meanings from Special Characters

If you include a special character on the command line and want the shell to interpret the character literally, you must remove the special meaning from the character. You can use the following characters to perform this task.

**** The backslash removes the special meaning from the character that follows it.

'...' Single quotation marks remove the special meaning from all characters within the pair of quotes.

"..." Double quotation marks remove the special meaning from all characters within the pair of quotes, with the exception of the dollar sign ($), the backquote (`), and the backslash (\).

4.8.1 Using the Backslash

The backslash removes the special meaning from the character that follows it. For example, if you type the command **echo ***, the shell will first perform filename substitution, replacing the asterisk with the names of the files in the current directory. It will then execute the **echo** command, which lists the filenames on your screen.

```
$ echo *  Return
buchanan letter_488 report1 report2
$
```

If you don't want the shell to perform filename substitution, precede the asterisk with a backslash. When the shell reads the command line, it will interpret the asterisk literally and ignore its special meaning. In this case, the **echo** command prints the asterisk.

```
$ echo \*  Return
*
$
```

The backslash is also useful when you are typing a command that requires more than one line on the screen. When you reach the right margin, type a backslash and press Return, then continue the command on the next line. The backslash tells the shell to ignore the Return.

```
$  grep 'meeting time' group.memo agenda \ Return
room.sched > time &  Return
```

4.8.2 Using Single Quotes

If you need to remove the special meaning from a string of characters, enclose the string in single quotation marks. When the shell interprets the command line, it will not assign any special meaning to the characters within the quotes.

For example, you might have a file named *lost* that contains this line:

```
Where is it???
```

If you want to use **grep** to locate this line, you could type the following command, enclosing the question marks in quotes.

```
$ grep '???' lost  Return
Where is it???
$
```

If you omit the quotation marks, the shell will try to perform filename substitution on the line, replacing the question marks with those filenames in your current directory that are three characters long.

Single quotes are also useful if an argument on the command line contains spaces. The shell interprets spaces on a command line as delimiters between the arguments of a command. When you place quotes around the argument, the shell ignores the special meaning attached to spaces.

For example, to use **grep** to find lines in the *baseball* file containing the words `The all`, you would type this command line:

```
$ grep 'The all' baseball  Return
The all star game started on time.
$
```

If you omitted the quotation marks, the shell would search for the word `The` in the *all* file (which doesn't exist) and in the *baseball* file.

```
$ grep The all baseball  Return
all: No such file or directory
baseball: The all star game started on time.
```

4.8.3 Using Double Quotes

Double quotation marks also remove special meanings from the characters included in the quotes. However, they do not remove the special meaning from backquotes (`), backslashes (\), or the dollar sign ($). In most cases, you can use single quotes to enclose a string of characters. However, if the string includes a single quotation mark, you will need to enclose the string in double quotes.

```
$ grep "He can't attend" seminar  Return
```

Interpreting Backquotes

You should also use double quotes if the string being quoted includes command substitution (indicated by backquotes) and you want the substitution to occur. The double quotes tell the shell to interpret the backquotes as special characters and to perform the command substitution.

The arguments to the following **echo** command are enclosed in double quotes. This tells the shell to ignore the special meaning of the single quote (') and the question mark (?). The double quotes also allow the command substitution to take place. The shell first performs the command substitution (substituting the output from the **who** pipeline) and then executes the **echo** command.

```
$ echo "Who's out there?  `who | wc -l` users"  Return
Who's out there?   143 users
$
```

Interpreting the Dollar Sign

A *variable* is a name that represents a string of characters. The shell uses a set of variables to define your environment. (These variables are described later in this chapter.) You can also create your own variables for use on the command line or in shell programs.

The dollar sign ($) is used for variable substitution. It tells the shell to replace the variable name following the dollar sign with its actual value. If you are quoting a string that includes variable substitution and you want the substitution to take place, enclose the string in double quotes.

For example, the variable **x** could have the value `* light`. You can use the **echo** command, followed by a dollar sign and the variable name, to print the value of the **x** variable. However, if you type the command as **echo $x**, the shell first substitutes the value of the **x** variable, `* light`. It then performs filename substitution and replaces the asterisk with the filenames in the current directory. When the **echo** command executes, it displays the filenames followed by the word `light`.

```
$ echo $x  Return
buchanan letter_488 report1 report2 light
$
```

If you enclose the variable in single quotes, the shell will ignore the special characters. It will not perform variable substitution or filename substitution, and the **echo** command will print the string you typed.

```
$ echo '$x'  Return
$x
$
```

When you enclose the string in double quotes, the dollar sign keeps its special meaning and the shell performs variable substitution. When the shell reads `* light`, the substituted value, it does not perform filename substitution since the double quotes remove the special meaning attached to the asterisk. The **echo** command then prints the value attached to the **x** variable.

```
$ echo "$x"  Return
* light
$
```

Interpreting the Backslash

The location of a backslash (\) within double quotes determines whether the shell recognizes its special meaning. If the backslash is the first character within the quotes, the shell ignores its special meaning.

```
$ echo "\ is called backslash"  Return
\ is called backslash
```

If the backslash is not the first character in quotes, the shell interprets the backslash with its special meaning. This allows you to use a backslash to remove special meaning from characters (backquotes, dollar signs, other backslashes, other double quotes) that would otherwise retain that meaning within the quotes.

In the following example, we enclosed the string that **echo** is to print in double quotation marks because the string includes variable substitution. Because we want **echo** to print the double quotes around **$x**, we typed a backslash in front of these quotation marks to remove their special meaning.

```
$ echo "\"$x\" moonlight"  Return
"* light" moonlight
```

4.9 The Bourne and Korn Shell Environment

This section describes features common to the Bourne and Korn shell environments. Features that are specific to the Korn shell are described in the section "Using the Korn Shell," later in this chapter. For information about the C shell, refer to "Using the C Shell," at the end of this chapter.

Each user on the system works in an environment that is set when the user logs in. The environment defines variables such as the command prompt that appears on the screen, the search path used to locate commands, and the name of your home directory. You can use the default environment set up for you, or you can tailor the environment to suit your individual needs.

When you log in, the shell first examines a file called */etc/profile*. The system administrator sets up this file, which performs tasks such as checking for mail. This file can include any commands or variables the system administrator wants to specify.

The shell then examines a file named *.profile* in your home directory. This file was created for you when your login account was set up and contains commands and variables that define your environment. If you want to modify your environment, you can add other commands and variables to your *.profile*.

Other Information

ksh(1), **sh**(1), **profile**(4)

4.9.1 Environment Variables

The variables that define your environment are called *environment variables*. They appear in uppercase letters, making them easy to recognize. When you log in, the shell and the **login** program assign values to many of these variables. Other environment variables are stored in files where programs needing those values can access them. The system administrator or the user can also assign values to some variables.

The shell passes the values assigned to environment variables on to each program you start up, including new copies of the shell program itself. This is called *exporting*.

Following are some of the environment variables used by the shell and other programs:

CDPATH	Defines the search path for the **cd** command. The shell uses this variable when you invoke the **cd** command without a full pathname.
EDITOR	Contains the name of an editor (typically **ed**, the line editor).
HOME	Contains the pathname of your login directory. It is the default argument for the **cd** command. If you do not specify a directory, **cd** will move you to the directory specified by this variable.
IFS	Specifies the characters the shell interprets as internal field separators (usually space, tab, and newline).
LOGNAME	Contains your login name.
LPDEST	Contains the name of your default destination printer for the LP print service.
MAIL	If this variable is set to the name of a file, the shell informs you when mail arrives in the file.
MAILCHECK	Specifies how often (in seconds) the shell checks for the arrival of mail in the files specified by **MAIL** or **MAILPATH**. The default value is 600 seconds (10 minutes). If you set this variable to zero, the shell will check for mail before each shell prompt displays.
MAILPATH	Can contain a list of filenames. If this variable is set, the shell informs you when mail arrives in any of the files. Typically you will use either the **MAIL** or the **MAILPATH** variable.
PARALLEL	Determines the number of commands that can run in parallel. Many operating system utilities such as **lint** and **make** use this variable to schedule commands for execution.
PATH	Lists directories that contain programs. When you issue a command, the shell searches these directories to locate the command.
PS1	Specifies the primary prompt string (default is $).
PS2	Specifies the secondary prompt string (default is >). The shell displays this prompt when a command requires more input from you.

SHELL	Contains the name of your shell program (for example, */bin/sh*). Programs such as the **ed** and **vi** editors use this variable to determine which shell program to start up when you issue a shell command while you are using the editor.
SHACCT	If this variable is set to the name of a file for which you have write permission, the shell will write an accounting record in that file whenever you use a shell procedure.
TERM	Defines the type of terminal you are using. Programs such as the **vi** screen editor use this variable to determine how to display text on your screen.
TERMINFO	Contains the name of the file storing information about your terminal.
TZ	Contains the timezone for your location.
VISUAL	Contains the name of an editor (typically **vi**, the screen editor).

To see a list of the variables set in your environment, type **set** and press Return. The following screen shows sample output from the **set** command.

```
$ set Return
HOME=/user1/sarah
IFS=

LOGNAME=sarah
MAIL=/usr/mail
MAILCHECK=600
PATH=:/bin:/usr/bin
PS1=$
PS2=>
SHELL=/bin/sh
TERM=5420
TZ=PST8PDT
```

You can use the **echo** command to display the value of a specific environment variable. To do this, type **echo** followed by ***$variable_name***. The dollar sign ($) is a special character that tells the shell to substitute the value of the variable. After the shell substitutes the appropriate value, it executes the **echo** command, which prints its arguments on the screen.

For example, the following command will display the value assigned to user *terri's* **HOME** variable:

```
echo $HOME Return
/user1/terri
```

4.9.2 Modifying Environment Variables

The shell allows you to assign new values to many environment variables. If you assign a value on the command line, the new value affects only that login session. If you want to assign a value permanently, include that environment variable in your *.profile* file.

To set an environment variable, type the name of the variable followed by an equal sign (=) and the new value. Do not insert spaces around the equal sign. For example, the **PS1** variable specifies the character used as the command prompt ($). If you want to change the value of the **PS1** variable to @, you would type the following line:

```
PS1='@ '
```

Because the new value for the **PS1** variable contains a space, we enclosed the value in single quotes. You must also use quotes if a value contains special characters.

When you assign a value to a variable, you must *export* the variable. This step makes the new value available to all other programs. To export a variable, type **export** followed by the name of the variable.

```
export PS1
```

Setting a Variable for the Current Login Session

To set a variable for the current login session only, assign the new value on the command line and then export the variable. The following screen shows the entries you would make to change the **PS1** variable to @ for the current login session.

```
$ PS1='@  '  Return
$ export PS1  Return
@
```

Setting a Variable in Your *.profile*

The environment variables included in your *.profile* are set whenever you log into the system. Typically, a *.profile* contains the **PATH** and **TERM** variables. You can also add other environment variables. (Use caution the first few times you edit your *.profile*. Create a copy of the file before making any changes to it. After editing *.profile* successfully, you can delete the copy.)

You can set a variable in your *.profile* the same way you set a variable on the command line. For example, to modify the **PS1** variable as described in the previous section, you would add this line to your *.profile*:

```
PS1='@ '
```

You must export the environment variables included in your *.profile* to make them available to other programs. You can export all of the variables on the same line. If you use this method, whenever you define a variable in your *.profile*, add the variable name to the **export** line.

```
export PATH TERM PS1
```

You can also include the **export** command on the line where you set the variable:

```
PS1='@ ';export PS1
```

If you want to return **PS1** to its original value, simply delete the line defining the **PS1** variable.

Setting the PATH Variable. Occasionally, you might need to modify the **PATH** environment variable. This variable stores the search path the system uses to locate and execute commands. To see the current value of your **PATH** variable, type the following command:

```
$ echo $PATH Return
.:/mylogin/bin:/bin:/usr/bin:/usr/lib
$
```

The colon (:) is a delimiter between pathnames. The dot in the first position represents the current directory. In our example, the system first looks for commands in the current directory, then in */mylogin/bin*, then in */bin*, then in */usr/bin*, and finally in */usr/lib*.

If you want the shell to search another directory, you must add that directory to your path. For example, if you are working on a project with several other people, you could set up a *bin* directory containing shell programs used only by your project members. The path to this directory might be */project1/bin*. To tell the system to search this directory, you would need to edit your *.profile*, adding the new directory to the end of your **PATH** variable.

```
PATH=":/mylogin/bin:/bin:/usr/lib:/project1/bin"
```

Setting the TERM Variable. If you log into the system over a line attached directly to the computer, the system determines the type of terminal you are using and sets your **TERM** variable. If you log in over a dial-up line, you need to tell the system the type of terminal you are using. You can specify this information when you log in, or you can set the **TERM** variable in your *.profile*.

To set your terminal type when you log in, type your login name at the login prompt, type a space, then type **TERM=***terminal_type*. Your system administrator can supply you with the appropriate *terminal_type*.

```
login: chris TERM=sqnt220 Return
```

To set your terminal type in your *.profile*, add a line in the following form to your file.

TERM=*terminal_type***;export TERM**

Executing *.profile*

When you add entries to your *.profile*, the entries do not take effect until you either log in again or execute the dot (.) command. This is because the system reads *.profile* only when you log in. The dot (.) command tells the system to reexecute *.profile*, making the entries effective in the current login session. If you make additions to *.profile*, type the following command to make the additions effective immediately:

```
$ ..profile Return
```

If you delete an entry from your *.profile*, the change will not take effect until you log in again. For example, your *.profile* might include this line:

```
LPDEST=lp3;export LPDEST
```

Deleting this line from *.profile* does not remove the **LPDEST** variable from your environment. Your print requests will continue to go to printer *lp3* until you either log in again or redefine the **LPDEST** variable.

4.9.3 Adding Commands to Your *.profile*

If you want particular commands to execute whenever you log into the system, you can add the commands to your *.profile*. For example, you could add the following **echo** command to your file:

```
echo Good Morning!
```

The next time you log in, this message will appear:

```
Good Morning!
$
```

You can add any commands you want to your *.profile*. Typically, *.profile* files include the **stty** command described in the next section.

Setting Terminal Options: stty

The **stty** command sets up your terminal. This command includes options that affect data transmission and determine how characters are treated. Your *.profile* probably includes an **stty** command to set up your terminal for your site.

To see the options set for your terminal, type **stty** and press Return. If your display includes a line similar to the following, your erase and kill keys have been reset from their default values.

```
erase = ^h; kill = ^x;
```

The character following `erase =` is your erase character; the character following `kill =` is your kill character. The caret represents the Ctrl key. (Typically, BackSpace is `^h`.)

The following paragraphs describe several **stty** options you can set, including your erase and kill keys.

–tabs This option preserves tabs when you are printing. It expands the tab setting to eight spaces, which is the default.

erase ^h This option changes the erase character from the default value (#) to BackSpace.

echoe If your terminal has a screen, this option erases characters from the screen as you erase them with BackSpace.

kill ^x This option changes the kill key from the default character (@) to Ctrl-X.

If you want to use these options, add the following line to your *.profile*:

stty –tabs erase ^h echoe kill ^x

Other Information

stty(1)

4.10 Using the Korn Shell

The Korn shell is similar to the Bourne shell, but it includes several additional features. This section tells how to start up the Korn shell and describes these Korn shell features:

- Editing the command line
- Using your command history to reexecute commands
- Using aliases to customize commands
- Using **cd** command options to move between directories
- Using a tilde (~) to represent a home directory
- Substituting command output as an argument

The Korn shell also includes job control, described in Chapter 7, and adds several shell programming features.

Other Information

ksh(1)

4.10.1 Invoking the Korn Shell

If the Korn shell is your default shell, it will start up automatically whenever you log into the system. If the Korn shell is not your default shell, you can start it up by typing **ksh** on the command line.

```
$ ksh Return
```

You will continue to see the dollar sign ($) shell prompt and the operating system commands will execute in the same manner as in the Bourne shell. However, you can also use the Korn shell features described in this section.

If you want to make the Korn shell your default shell, contact your system administrator, who can modify your login account. The Korn shell will then start up whenever you log into the system.

4.10.2 The Korn Shell Environment

The Korn shell includes several additional environment variables. The shell defines some of these variables; you can assign values to the other variables in your *.profile*. Following are a few of the Korn shell environment variables:

PWD Stores the name of your current working directory.

OLDPWD Stores the name of your previous working directory.

ENV Stores the name of a startup file that can contain aliases, functions, and other commands. When the Korn shell is invoked, it reads this file. You can call the file whatever you want. If you named the file *.kshrc*, you would add this line to your *.profile*:

```
ENV=${HOME}/.kshrc
```

FCEDIT Defines the editor used by the **fc** commands described later in this section. The default is **ed**.

HISTFILE Contains the name of the history file used to store commands. The default is *.history* in your home directory.

HISTSIZE Specifies the number of commands you can access in the history file. The default is 128 commands.

_ (underscore) Contains the last argument from your previous command.

The Korn shell assigns a job number to each command you issue, allowing you to reissue or edit commands easily. You can modify the **PS1** variable to print the current job number before the shell prompt ($). To do this, add the following line to your *.profile*:

```
PS1='! $ '; export PS1
```

When this change takes effect, the current job number will print before the shell prompt. In the following example, the shell will assign job number 122 to your next command.

```
122 $
```

4.10.3 Editing the Command Line

The Korn shell allows you to use **vi** or **emacs** screen editor commands to edit the command line you are typing. This section tells how to use **vi** commands. If you are an **emacs** user, refer to **ksh**(1) for information about editing with **emacs** commands.

If you want to use **vi** to edit the command line, set the **VISUAL** environment variable to **vi** and export it in your *.profile*.

```
VISUAL=vi; export VISUAL
```

When this variable is set, the system goes into **vi** input mode whenever you type a command. If you want to edit the command line, press Esc to place the system in **vi** command mode. You can now use **vi** commands to move the cursor and to delete, replace, and substitute characters. If you want to add characters, use the **a** or **i** command to enter input mode, then type the characters. When you finish adding characters, press Esc to return to command mode.

When you are ready to execute the command, press Return.

Following is a brief list of **vi** commands useful for editing a command line. Chapter 9 describes the **vi** commands in detail.

h	Moves cursor left
l	Moves cursor right
0 (zero)	Moves cursor to beginning of line
$	Moves cursor to end of line
w	Moves to beginning of next word
b	Moves to beginning of previous word
x	Deletes character
rc	Replaces current character with character *c*
cw	Changes word
dw	Deletes word
U	Restores line to original state

4.10.4 Command History

The Korn shell stores the commands you enter in a file called *.history* in your home directory. By default, the file stores your last 128 commands. You can view, edit, or reexecute these commands at any time.

You can use **vi** to scroll through the list of commands in your *.history* file. To display the previous command, first press Esc, then type either **k** or a hyphen (–). You can continue pressing **k** or a hyphen (–) to scroll back through your list of commands. To scroll forward, use either **j** or a plus sign (+).

If you want to edit a command line, use **vi** as described in the previous section. When you press Return, the command executes.

Displaying Command History

The **history** command lists your last 16 commands on the screen. If you want to see a different number of commands, follow **history** with a minus sign and the number. For example, to see your last 20 commands, you would type **history –20**; to see your last 5 commands, you would type **history –5**.

The following screen shows sample output from the **history** command.

```
$ history  Return
[127] date
[128] pwd
[129] cd budget
[130] ls
[131] vi april
[132] lp -c -n2 april
[133] cp april ../q2
[134] lpstat
[135] cd ../accts
[136] grep delinquent accounts > del_acct &
[137] mailx
[138] pg del_acct
[139] vi chapter10
[140] lp -n2 chapter10
[141] date
[142] rm chapter10
```

Reexecuting Commands

The **r** command reexecutes commands. To reexecute your previous command, type **r** and press Return. If you want to execute a different command, type **r** followed by either the number assigned to the command or a few characters from the command. Both of the following lines will reexecute command 136 on the previous screen.

```
$ r 136 Return
```

```
$ r grep Return
```

If the command you reexecute runs in the background, the system will display a job number followed by the command. If the command runs in the foreground, the command and its output will appear.

You can also use the **r** command to substitute characters on the command line. To do this, type the original characters followed by an equal sign (=) and the new characters. Then type either the number assigned to the original command or a few characters from the command. For example, to modify the command **rm chapter10** (command 142 on the previous screen) to delete *chapter12*, you would type this command:

```
$ r 10=12 142 Return
```

Editing Command Lines

You can also use the **fc** command to edit a previous command line. The **fc** command sends the command line you specify through a text editor, allowing you to make the necessary changes. The default editor is **ed**. If you prefer to use **vi**, set the **FCEDIT** variable in your *.profile*.

```
FCEDIT=vi; export FCEDIT
```

To edit a command, type **fc** followed by the job number. For example, the following command line invokes the **vi** editor on command 132.

```
$ fc 132 Return
```

The shell places the command line in a temporary file where you can use **vi** to edit it. When you leave the editor, the shell executes the command.

You can also edit a series of commands. To do this, type **fc** followed by the first and last commands in the series. For example, to edit commands 133 through 137, you would type this command:

```
$ fc 133 137 Return
```

The shell will place the commands you specified in a temporary file, where you can use **vi** to edit them. When you leave the editor, all of the commands will execute.

Searching for a Command

You can also search through the history file for a specific command. Press Esc, then type a slash followed by a few characters from the command. The shell will then locate your last command including those characters. The characters can be in any position on the command line. You can use **vi** to edit the command, or you can press Return to execute it. For example, if you type the following line, the shell will locate your last **sort** command.

```
$ Esc  /sort  Return
$ sort mail.list > mailing &
```

If you want to locate the next command containing the search pattern, press **n** to continue searching backward through the history file. If you want to search forward through the history file, press **N**.

4.10.5 Creating Aliases

An alias is a shorthand notation that allows you to customize commands. The Korn shell maintains a list of aliases that represent commands or shell programs. When the shell reads a command line, it checks the command name against the list of aliases. If the command name is on the list, the shell substitutes the value of the alias for the command name.

Several aliases are already defined in the shell. To see a list of these aliases, type **alias** and press Return. You can also use the **alias** command to create your own aliases. If you create an alias on the command line, it will be in effect during the current session only. If you want to make the alias permanent, add it to the file defined by your **ENV** environment variable.

The **alias** command has this format:

alias *name=value*

The *name* can begin with any character other than a special character. All other characters in the name must be numbers or letters. The *value* can be any valid shell program.

Aliases are frequently used to redefine operating system commands. For example, the following alias redefines the **rm** command.

```
alias rm='rm -i'
```

When you issue the **rm** command, it will now execute as **rm –i**.

The **unalias** command removes an alias for the current login session. For example, the following command removes the **rm** alias from the alias list.

```
$ unalias rm  Return
```

4.10.6 The cd Command

The **cd** command has some additional features under the Korn shell. If you change directories and then want to return to your previous working directory, type **cd** followed by a minus sign (–).

```
$ pwd  Return
/user1/puma/letters
$ cd; pwd  Return
/user1/puma
$ cd -  Return
/user1/puma/letters
$
```

You can also use substitution to specify a directory name. To do this, follow the **cd** command with the name of the current directory and the name of the directory you want to move to. For example, the **cd** command on the following screen changes directory from */user1/puma/letters* to *user1/puma/schedules*.

```
$ pwd  Return
/user1/puma/letters
$ cd letters schedules  Return
/user1/puma/schedules
```

4.10.7 Using a Tilde to Represent a Home Directory

If a word on the command line begins with a tilde (~), the Korn shell will determine if the word meets any of the following conditions and then make the appropriate substitution.

- If the tilde is the only character in the word or is followed by a slash, the shell substitutes the value of your **HOME** environment variable (your home directory).

  ```
  $ echo ~  Return
  /user1/puma

  $ cd ~/letters  Return
  $ pwd  Return
  /user1/puma/letters
  ```

- If a login name follows the tilde, the shell substitutes the home directory associated with that login name. For example, the following command copies the *letter* file from user *wildcat's* home directory to user *puma's* home directory.

  ```
  $ cp ~wildcat/letter ~puma  Return
  ```

- If a plus sign (+) follows the tilde, the shell substitutes the value of your **PWD** environment variable (your current working directory). If a minus sign (–) follows the tilde, the shell substitutes the value of your **OLDPWD** environment variable (your previous working directory).

```
$ pwd  Return
/user1/puma/letters
$ cd; pwd  Return
/user1/puma
$ echo ~+  Return
/user1/puma
$ echo ~-  Return
/user1/puma/letters
```

4.10.8 Command Substitution

Command substitution tells the shell to use the output from one command as an argument for another command. The Bourne shell uses backquotes to indicate the command to substitute. (For more information, see "Using Command Output as an Argument," earlier in this chapter.)

The Korn shell can interpret backquotes correctly but it also offers another way to perform command substitution. In place of the backquotes, use parentheses to enclose the command whose output is to be substituted and precede the opening parenthesis with a dollar sign ($).

The shell first evaluates the command in the parentheses. It then substitutes the command output as if it were enclosed in double quotes. In the following example, the shell substitutes the output from the **date** and **cut** pipeline as the argument to **echo**. The command then prints the current time.

```
$ echo $(date | cut -c12-19)  Return
10:40:23
$
```

If you use this notation for some other purpose and don't want command substitution to occur, precede the dollar sign with a backslash (\$).

4.11 Using the C Shell

The C shell interprets your commands in the same way as the Bourne shell. It also includes several additional features. This section tells how to start up the C shell, discusses the C shell environment, and describes these C shell features:

- Using aliases to customize commands
- Using your command history to reexecute commands
- Using a tilde to represent a home directory
- Using the **pushd** and **popd** commands to change directories

The C shell also includes job control, described in Chapter 7, which allows you to stop or kill jobs and to move them to the foreground or background.

Other Information

csh(1)

4.11.1 Invoking the C Shell

If the C shell is your default shell, it will start up automatically whenever you log into the system. If it is not your default shell, you can start it up by typing **csh** on the command line.

```
$ csh  Return
%
```

You will now see a percent sign (%) as the shell prompt. The operating system commands will execute in the same manner as in the Bourne shell but you can also use the C shell features described in this section.

If you want to make the C shell your default shell, contact your system administrator who can modify your login account. The C shell will then start up whenever you log into the system.

4.11.2 The C Shell Environment

Each user on the system works in an environment set when the user logs in. The environment defines values such as the command prompt that appears on the screen, the search path used to locate commands, and the name of your home directory. You can use the default environment set up for you, or you can tailor the environment to suit your individual needs.

When you log in, the shell first examines a file called */etc/login*. The system administrator sets up this file, which performs tasks such as checking for mail. This file can include any commands or variables the system administrator wants to specify.

The shell then examines two files in your home directory: *.login* and *.cshrc*. These files were created for you when your login account was set up. The *.login* file contains variables and commands that are executed when you log in. The *.cshrc* file includes variables and commands that affect the C shell itself. These variables are set both when you log in and when a new shell program is started up.

4.11.3 Environment Variables

The variables that define your environment are called *environment variables*. These variables appear in uppercase letters, making them easy to recognize. When you log in, the shell and the **login** program assign values to many of these variables. Other environment variables are stored in files where programs needing those values can access them. The system administrator or the user can also assign values to some variables. The shell passes the values assigned to environment variables on to each program you start up.

Following are some of the environment variables used by the shell and other programs:

EDITOR Contains the name of an editor (typically **ed**, the line editor).

HOME Contains the pathname of your login directory. It is the default argument for the **cd** command. If you do not specify a directory, **cd** will move you to the directory specified by this variable.

LOGNAME Contains your login name.

LPDEST Contains the name of your default destination printer for the LP print service.

MAIL Contains the name of the file used to store your mail. The shell informs you when mail arrives in the file.

PARALLEL Determines the number of commands that can run in parallel. Many operating system utilities such as **lint** and **make** use this variable to schedule commands for execution.

PATH Lists directories that contain programs. When you issue a command, the shell searches these directories to locate the command.

SHELL Contains the name of your shell program (for example, **/bin/csh**). Programs such as the **ed** and **vi** editors use this variable to determine which shell to start up when you issue a shell command.

TERM Defines the type of terminal you are using. Programs such as the **vi** screen editor use this variable to determine how to display text on your screen.

TERMINFO Contains the name of the file storing information about your terminal.

TZ Contains the timezone for your location.

USER Contains your login name.

VISUAL Contains the name of an editor (typically **vi**, the screen editor).

4.11.4 C Shell Variables

The C shell also includes a set of variables that appear in lowercase letters. Some of these variables take a specific value; other variables are either enabled or disabled. Following are several of the commonly used C shell variables. Notice that many of these variables correspond to an environment variable, such as **path** and **PATH**. When the C shell variable is set, the corresponding environment variable is also set.

cdpath Defines the search path for the **cd** command. The shell uses this variable when you invoke the **cd** command without a full pathname.

cwd Contains the name of your current directory.

history	Specifies the number of commands saved in your history list.
home	Contains the name of your home directory.
ignoreeof	If this variable is set, you cannot type Ctrl-D to log out of the operating system. Instead, you must type **logout**.
noclobber	If this variable is set, the shell will not overwrite an existing file when you use the greater-than sign (>) to redirect output to a file. Also, if you use two greater-than signs (>>) to append output to a file (as in the **cat** command), the shell will not create a new file for the output.
path	Lists directories to be searched for commands.
prompt	Defines the C shell command prompt (typically `%`).
shell	Contains the full pathname of your shell program.
term	Stores your terminal type.

4.11.5 Displaying the Value of a Variable

To see a list of the C shell variables set for you, type **set** and press Return. Following is some sample output from the **set** command.

```
% set Return
argv     ()
cwd      /user1/puma
history20
home     /user1/puma
ignoreeof
noclobber
path     (. /user1/puma/bin /bin /usr/bin usr/lib)
prompt   %
shell    /bin/csh
status   0
term     wy75
```

You can use the **echo** command to display the value of a specific variable (both C shell variables and environment variables). To do this, type **echo** followed by a dollar sign (**$**) and the name of the variable. The dollar sign (**$**) is a special character that tells the shell to substitute the value of the variable. After the shell substitutes the appropriate value, it executes the **echo** command, which prints its arguments on the screen.

For example, the following command prints the value assigned to user *terri's* **HOME** variable:

```
% echo $HOME Return
/user1/terri
```

4.11.6 Setting Variables

The shell, the **login** program, and other programs set many of the environment and C shell variables for you. However, you can assign values or enable or disable several variables. The **setenv** command is used to set environment variables; the **set** command sets C shell variables.

You can set variables either on the command line or in your *.login* or *.cshrc* file. If you set a variable on the command line, it will be in effect during the current login session only. To set a variable permanently, add it to your *.login* or *.cshrc* file.

Setting Environment Variables

The shell allows you to assign new values to many environment variables. To do this, type **setenv** followed by the name of the variable and the new value. For example, to set the **VISUAL** environment variable to the **vi** editor, you would type this line:

```
setenv VISUAL vi
```

Setting C Shell Variables

The shell assigns values to many variables, such as **home** and **shell**. However, you can use the **set** command to set some variables yourself. If the variable takes a value, type a command in this form:

```
set variable=value
```

For example, to assign a value of 20 to the **history** variable, you would type this command:

```
set history=20
```

If the value includes spaces, enclose the value in quotation marks.

If you want to enable a variable that does not take a value, simply follow the **set** command with the name of the variable. For example, to enable the **ignoreeof** variable, you would type this line:

```
set ignoreeof
```

Adding Variables to Your *.login* or *.cshrc* File

The variables included in your *.login* file are set whenever you log into the system. Typically, a *.login* file includes variables that affect the whole system, such as variables that set up your terminal.

The *.cshrc* file includes variables that affect the C shell itself. These variables are set when you log into the system and when you create a new copy of the shell program. For example, if you are working in the **vi** text editor, you can leave the editor temporarily and start up a new copy of the shell program in order to execute an operating system command. When the new shell program is started up, the system reads your *.cshrc* file and sets the variables it contains for the new shell.

Use caution the first few times you edit your *.login* or *.cshrc* file. Create a copy of the file before making any changes to it. After editing the file successfully, you can delete the copy.

Following are a few variables that you might want to modify.

Changing Your Path. The **path** variable stores the search path the system uses to locate and execute commands. To see the current value of your **path** variable, type the following command:

```
% echo $path (Return)
. /mylogin/bin /bin /usr/bin /usr/lib
%
```

The dot represents the current directory. In our example, the system first looks for commands in the current directory, then in */mylogin/bin*, then in */bin*, then in */usr/bin*, and finally in */usr/lib*.

If you want the shell to search another directory, you must add that directory to your path. For example, if you are working on a project with several other people, you could set up a *bin* directory containing programs used only by your project members. The path to this directory might be */project1/bin*. To tell the system to search this directory, first locate your **path** variable (it will be in either your *.login* or *.cshrc* file). Then add the new directory to your path.

```
set path=(. mylogin/bin /bin /usr/lib /project1/bin )
```

When you log into the operating system, the shell creates a table containing the locations of commands found in your path. To tell the shell to recreate this table, issue the **rehash** command whenever you change your **path** variable. You should also type **rehash** if a new command is added to a directory in your path.

```
% rehash (Return)
```

If you don't issue the **rehash** command, changes to your **path** variable will not take effect until the next time you log in.

Setting the TERM Variable

If you log into the system over a line attached directly to the computer, the system determines the type of terminal you are using and sets your **TERM** variable. If you log in over a dial-up line, you need to tell the system the type of terminal you are using. You can specify this information when you log in, or you can set the **TERM** variable in your *.login* file.

To set your terminal type when you log in, type your login name at the login prompt, type a space, then type **TERM=***terminal_type*. Your system administrator can supply you with the appropriate *terminal_type*.

```
login: chris TERM=sqnt220 (Return)
```

To set your terminal type in your *.login*, add a line in the following form to your file.

setenv TERM *terminal_type*

Changing Your Command Prompt

The C shell assigns a number to each command you issue. You can modify the **prompt** variable to print the current number before the shell prompt (%). To do this, add the following line to your *.cshrc* file:

```
set prompt='! % '
```

The quotation marks are necessary because the prompt includes spaces. After this change takes effect, the current number will print before the shell prompt. In the following example, the shell will assign number 128 to your next command.

```
128 %
```

4.11.7 Executing *.login* and *.cshrc*

When you add entries to your *.login* or *.cshrc* files, the entries do not take effect until you either log in again or execute the **source** command. This command tells the system to reexecute the file, making the entries effective immediately.

% **source .login** Return

% **source .cshrc** Return

If you remove entries from your *.login* or *.cshrc* files, the changes will not take effect until you log in again. For example, your *.login* file might include this line:

```
setenv LPDEST lp3
```

Deleting this line from *.login* will not remove the **LPDEST** variable from your environment. Your print requests will continue to go to printer *lp3* until you either log in again or redefine the **LPDEST** variable.

4.11.8 Adding Commands to Your *.login* File

If you want particular commands to execute whenever you log into the system, you can add the commands to your *.login* file. For example, you could add the following **echo** command to your file:

```
echo Good Morning
```

The next time you log in, this message will appear:

```
Good Morning
%
```

You can add any commands you want to your *.login*. Typically, *.login* files include the **stty** command, which is described in the next section.

Setting Terminal Options: stty

The **stty** command sets up your terminal. This command includes options that affect data transmission and determine how characters are treated. Your *.login* file probably includes an **stty** command that sets up your terminal for your site.

To see the options set for your terminal, type **stty** and press Return. If your display includes a line similar to the following, your erase and kill keys have been reset from their default values.

```
erase = ^h, kill = ^x
```

The character following `erase =` is your erase character; the character following `kill =` is your kill character. The caret represents the Ctrl key. (Typically, BackSpace is `^h`.)

The following paragraphs describe several **stty** options you can set, including your erase and kill keys.

–tabs This option preserves tabs when you are printing. It expands the tab setting to eight spaces, which is the default.

erase ^h This option changes the erase character from the default value (#) to BackSpace.

echoe If your terminal has a screen, this option erases characters from the screen as you erase them with BackSpace.

kill ^x This option changes the kill key from the default character (@) to Ctrl-X.

If you want to use these options, add the following line to your *.login*:

stty -tabs erase ^h echoe kill ^x

Other Information

stty(1)

4.11.9 Creating Aliases

An alias is a shorthand notation that allows you to customize commands. When the shell reads a command line, it checks the command name against the aliases you have created. If the command name is an alias, the shell substitutes the value of the alias for the command name.

To see a list of the aliases defined in your environment, type **alias** and press Return. You can also use the **alias** command to create your own aliases. If you create an alias on the command line, it will be in effect during the current session only. If you want to make the alias permanent, add it to your *.cshrc* file.

The **alias** command has this format:

alias *name value*

The *name* can begin with any character other than a special character. All other characters in the name must be numbers or letters. The *value* can be any valid shell program.

Aliases are frequently used to redefine operating system commands. For example, the following alias redefines the **rm** command.

```
alias rm 'rm -i'
```

When you issue the **rm** command, it will now execute as **rm -i**.

The **unalias** command removes an alias for the current login session. For example, the following command removes the **rm** alias.

```
% unalias rm Return
```

4.11.10 Command History

The C shell numbers the commands you enter during a login session. The shell also stores a specific number of these commands, allowing you to view or reexecute them at any time. These commands are called your *command list*. The number of commands the shell stores is determined by the **history** variable, which you can set in your *.cshrc* file. For example, to store the last 15 commands in your command list, you would add this line to your *.cshrc* file:

```
set history=15
```

To view your command list, type **history** and press Return. The following screen shows sample output from the **history** command. The commands are preceded by the numbers the shell assigned to them.

```
% history  Return
[128] pwd
[129] cd budget
[130] ls
[131] vi april
[132] lp -c -n2 april
[133] cp april ../q2
[134] lpstat
[135] cd ../accts
[136] grep delinquent accounts > del_acct &
[137] mailx
[138] pg del_acct
[139] vi chapter10
[140] lp -n2 chapter10
[141] date
[142] rm chapter10
```

Reexecuting Commands

The ! command reexecutes commands from your command list. To reexecute your previous command, type !! and press Return. If you want to execute a different command, type ! followed either by the number assigned to the command or by a few characters from the command. Both of the following lines will reexecute command 136 on the previous screen.

```
% !136  Return
```

```
% !grep  Return
```

If the command you reexecute runs in the background, the system will print the command and the number assigned to it. If the command runs in the foreground, the command and its output will appear on the screen.

Searching for a Command

You can also search through your command history to locate a specific command. First type an exclamation point, then type the string you are looking for, enclosed in question marks. For example, to search for your last command affecting the *del_accts* file, you could type this command:

```
% !?del?  Return
pg del_acct
```

Modifying Your Last Command

You can also modify the last command in your command list. The characters **!$** tell the shell to use the last argument from the previous command as the argument for the current command. For example, if your last command was **cat testfile**, the following line would tell the shell to execute the **pr** command on *testfile*.

```
% pr !$  Return
pr testfile
```

The characters **!*** tell the shell to use all arguments from the previous command as arguments for the current command. If your last command was **pg letter memo**, you could type the following line to tell the shell to run the **grep** command on the *letter* and *memo* files.

```
% grep February !*  Return
grep February letter memo
```

You can also follow the exclamation point with a number to indicate a specific argument to include in the current command. The first argument is 1, the second argument is 2, and so on. Command options are counted as an argument. When you use this method, type a colon (:) between the exclamation mark and the number. For example, your previous command might be **ls –l file1 file2 file3**. To run the **pg** command on *file2* (the third argument), you would type this command:

```
% pg !:3  Return
pg file2
```

Substituting Text in a Command

If you type a command line incorrectly and then press Return, an error message will appear. The C shell provides text substitution, allowing you to correct the command line. To do this, type a caret (^) followed by the text to be replaced. Then type another caret and the new text. The following example uses this method to correct a typing error on the command line.

```
% dtae  Return
dtae: Command not found.
% ^ta^at  Return
Wed Oct 19 15:34:09 PDT 1988
```

After we performed text substitution to correct the spelling of the **date** command, the shell executed the command.

4.11.11 Using a Tilde (~) to Represent a Home Directory

If a word on the command line begins with a tilde (~), the C shell will determine if the word meets either of the following conditions and then make the appropriate substitution.

- If the tilde is the only character in the word or is followed by a slash, the shell substitutes the name of your home directory.

```
% echo ~  Return
/user1/puma

% cd ~/letters  Return
% pwd  Return
/user1/puma/letters
```

- If a login name follows the tilde, the shell substitutes the home directory associated with that login name. For example, the following command copies the *agenda* file from user *wildcat's* home directory to user *puma's* home directory.

```
% cp ~wildcat/agenda ~puma  Return
```

4.11.12 Changing Directories

The C shell includes two commands that allow you to change directories without issuing the **cd** command. These commands are useful if you are moving to another directory and know that you will want to return to your original directory.

The **pushd** command tells the system to remember the name of your current directory and to move you to the directory you specify. For example, if you are working in */user1/puma/letters* and want to move to */user1/puma/project/specs*, you could type the following command. The system responds with the names of the directory you moved to and the original directory.

```
% pushd ../project/specs  Return
~/project/specs ~letters
% pwd  Return
/user1/puma/project/specs
```

When you are ready to return to your original directory, type **popd**. The name of the original directory then appears on the screen.

```
% popd  Return
~/letters
% pwd  Return
/user1/puma/letters
```

Chapter 5
Online Resources

5.1 Introduction 5-1

5.2 Online Man Pages 5-1

5.3 The help Command 5-3

5.4 ASSIST 5-6

Using ASSIST for the First Time 5-7

The ASSIST Menus 5-8

ASSIST Commands 5-9

Menu Layout 5-11

Displaying Pop-Up Help 5-12

Command Forms 5-14

Accessing Command Forms 5-14

Command Form Layout 5-15

Displaying Pop-Up Help 5-18

Moving the Cursor 5-20

Entering Input 5-20

Editing a Field 5-21

Validating Data 5-21

Executing a Command Line 5-22

Saving a Command Line 5-22

The ASSIST Walkthrus 5-23

The Pop-Up Menu 5-24

Chapter 5
Online Resources

5.1 Introduction

The operating system includes the following online resources:

- Copies of the man pages contained in the *Reference Manual*
- A **help** command that defines operating system terms and describes operating system commands
- An ASSIST program that helps you create command lines

This chapter tells how to use each of these features.

5.2 Online Man Pages

The man pages included in the *Reference Manual* are also stored online. You can use the **man** command to view them on your screen. To do this, type **man** followed by the name of the page you want to look at. The **man** command begins searching for the name you specified in section 1. If the name occurs in more than one section (such as **write**(1) and **write**(2)), **man** prints the first man page it locates with that name (in this case, **write**(1)).

```
$ man write  Return
```

You can also specify a section number on the command line. (The section number can include a trailing character such as **1g** or **1m**.) For example, to view the **write**(2) man page, you would type this command:

```
$ man 2 write  Return
```

If a man page contains more than one screen of text, **man** uses the **pg** command to paginate the text. The directions on the screen tell you how to use **pg** to view the remainder of the man page. The directions also tell you how to leave the man page display.

The **-k** option to the **man** command can locate a man page for you. Type **man -k**, then type a keyword that describes the page you are looking for. The system then lists all man pages containing that keyword in their description. The **apropos** command is equivalent to **man -k**.

```
$ man -k search
grep, egrep, fgrep (1) - search a file for a pattern
```

The **-f** option can locate a man page that describes a particular file. To use this option, type **man -f** followed by the filename. The system then lists all man pages containing that filename in their description. The **whatis** command is equivalent to **man -f**.

```
$ man -f passwd
passwd (1)          - change login information
passwd (4)          - password file
```

Other Information

apropos(1), **pg**(1), **man**(1), **whatis**(1)

5.3 The help Command

The **help** command describes many operating system commands and contains examples of their use. It also includes a glossary that defines operating system concepts. To use this command, type **help** and press Return. The following menu then appears on your screen.

```
help:   UNIX System On-Line Help

choices            description

   s               starter: general information

   l               locate: find a command with keywords

   u               usage: information about commands

   g               glossary: definitions of terms


   r               Redirect to a file or a command
   q               Quit


Enter choice >
```

Four of the choices on this menu (**starter**, **locate**, **usage**, and **glossary**) are interactive menu programs. To select one of these programs, type the corresponding letter at the `Enter choice >` prompt on the bottom of the screen. For example, to select the glossary menu, you would type **g** and press Return.

The **starter**, **locate**, **usage**, and **glossary** programs can also be called from the shell prompt. For example, you could use either of the following command lines to call the **glossary** program.

```
$ help glossary  Return

$ glossary  Return
```

The **r** option is available on all **help** screens. It allows you to redirect the text on the screen to a file or to pipe it to a command. For example, you might want to save a glossary definition or send it to a printer. To redirect the contents of a **help** screen to a file, type **r** and press Return. At the next prompt, type **>** followed by the filename. To redirect a **help** screen to a command, type **r** and press Return. Then, at the next prompt, type **|** followed by the command.

You can leave the **help** programs at any time by typing **q** and then pressing Return. The shell prompt then reappears on your screen.

Following are descriptions of each **help** program.

starter

The **starter** program displays general information about a specific topic. When you select this program, a list of topics appears on your screen. To see information about one of these topics, type the corresponding letter at the `Enter choice >` prompt and press Return.

For example, if you type **c** and press Return, a list of common operating system terms and commands will print on your screen. You cannot access the descriptions of these terms and commands from the **starter** program. The list is simply a pointer to commands and terms you might want to learn about first.

When you make a selection from the **starter** menu, a `Choices:` line appears at the bottom of the screen, followed by several options. Type the letter corresponding to the action you want to take, then press Return. For example, you could type **h** to go back to the **help** menu or **q** to exit **starter** and return to the shell.

locate

The **locate** program finds operating system commands related to a keyword that you specify. To use this command, type **k** at the `Enter choice >` prompt and press Return. This prompt then appears on the screen:

```
Enter keywords separated by blanks >
```

Type one or more keywords and press Return. For example, if you are looking for commands related to removing files, you could type **remove** as the keyword. The program will then print a list of commands related to that keyword.

The `Choices:` line at the bottom of the screen allows you to see additional information about any of the commands on the list. Simply type the name of the command and press Return. You can then ask for a description, an example, or a list of options. (The discussion of the **usage** program tells how to use this feature.) You can also specify additional keywords or leave the **locate** program.

usage

The **usage** program provides a description, examples, and a list of options for many operating system commands. To see a list of the commands included in the **usage** program, type **p** at the `Enter choice >` prompt and press Return.

If you want to see information about a specific command, type the name of the command at the `Enter choice >` prompt and press Return. This prompt then appears on your screen:

```
Enter d (description), e (examples), or o (options) >
```

Type the letter corresponding to the feature you want to see and press Return. The `choices:` line that appears at the bottom of the screen allows you to select the remaining information about the command, to specify another command, or to leave the **usage** program.

glossary

The **glossary** program defines many operating system terms. When you select this program, a list of terms appears on your screen. When you type one of these terms at the `Enter choice >` prompt, its definition appears on the screen. Be sure to type the term exactly as it appears on the list.

The `choices` line at the bottom of the screen allows you to ask for another glossary definition or to leave the **glossary** program.

Other Information

glossary(1), **help**(1), **locate**(1), **starter**(1), **usage**(1)

5.4 ASSIST

ASSIST is a menu interface that introduces you to the operating system, categorizes operating system commands by function, and helps you build and execute command lines.

ASSIST does not hide the operating system from you. Instead, it is a learn-while-doing environment. ASSIST includes these features:

Menus
The menus list groups of related commands that you can execute through the ASSIST program. Some menus also describe other commands that are related to the menu topic but cannot be executed through ASSIST.

Command forms
The command forms correspond to operating system commands. Each form contains questions that allow you to specify the options and arguments you want to apply to the command. As you answer the questions, the command line you are building appears at the bottom of the screen.

Pop-Up Menu
The Pop-Up Menu can be invoked from the ASSIST menus and command forms. It gives you direct access to ASSIST features, allows you to change directories, includes a keyword-based command search capability, and allows you to select the operating system and ASSIST walkthrus.

Pop-up help
Pop-up help is available on each menu and command form. The menu pop-up help describes the menu and each item it contains. On a command form, the pop-up help describes the command and each question on the form.

Walkthrus
Walkthrus are introductory programs that introduce you to the operating system, the ASSIST program, and the **vi** editor. The walkthrus are self-explanatory.

5.4.1 Using ASSIST for the First Time

To start up the ASSIST program, type **assist** at the shell prompt:

```
$ assist Return
```

The first time you use ASSIST, it does some preliminary checks and verifications to ensure that it can work effectively with your terminal. ASSIST uses a setup module to determine the type of terminal you are using and to find out if it has special capabilities such as highlighting, function keys, and an alternate character set.

To perform these checks, the setup module displays a series of test patterns and asks questions about what you see on your screen. The setup initialization screens are self-explanatory. The setup module stores the information it obtains about your terminal in a file called *.assistrc*. (If ASSIST cannot determine your terminal type, it will not function properly on your terminal. Contact your system administrator for assistance.)

During the setup procedure, the setup module determines if the function keys on your terminal will work with ASSIST. To make this check, the module asks you to press F8. If nothing happens, you will need to press any other key. The setup module will not do anything until you press another key.

If you press F8 and the message `Sorry; got something other than f8` appears, the system did not understand the signals from your terminal's function keys. If you do not have working function keys, this does not mean that you cannot use ASSIST. You can accomplish the same actions by using control characters instead of function keys.

If your terminal has programmable function keys that are programmed to something other than the default definition, ASSIST won't recognize the keys as working properly. For example, the default definition for your F1 function key might be F1=ESC[A. If you redefined the F1 function key as `F1=pwd; ls -l`, ASSIST won't recognize your definition.

To set up your terminal so programs such as ASSIST can recognize your function keys, you can add the line **tput init** to your *.profile* (Bourne and Korn shell users) or your *.login* file (C shell users). However, on some terminals this line might erase what you have programmed into the function keys. (Contact your system administrator if you need assistance.)

If you want to rerun the setup module at a later time, type this command:

```
$ assist -s (Return)
```

While you are in the setup module, you can redefine your terminal type, identify your terminal's name, and display introductory information about ASSIST.

5.4.2 The ASSIST Menus

After ASSIST has been set up for your terminal, the Top Menu will appear whenever you issue the **assist** command. This menu lists several second-level menus categorized by operating system command functions. For example, commands for comparing files are on one menu and programming commands are on another menu. A few of these menus have third-level menus.

```
CURRENT MENU: TOP  [ACCESSED DIRECTLY FROM OPERATING SYSTEM]  HELP: ^A
------------------------------------------------------------------------

             >Batch, Background, and Execution Control
              Compare Files
              Create, Copy, Rename, and Remove Files
              Directory-Related Commands
              Edit, Cut, and Paste Files

              Electronic Mail and Networking
              Find Files
              Format Text and Hardcopy Output
              Look Inside Files and Directories
              Permissions of Files and Directories

              Program
              Set Up User and Terminal Environments
              Sort, Transform, and Archive Files
              Statistics of Files, Directories, and Disk Blocks
              System, Terminal, and General Information
```

The greater-than sign (>) indicates the position of the cursor. To move the cursor on this menu, use the following keys:

Ctrl-P or UpArrow	Moves the cursor up
Return or DownArrow	Moves the cursor down

You can also go directly to an item by typing the first letter of that item. For example, to move to the Permissions of Files and Directories Menu, you would type **p**. If there is more than one item beginning with the letter you typed, the cursor moves to the first of those items. In our example, when you type **p**, the cursor moves to `Permissions of Files and Directories`. If you type **p** again, the cursor moves down to `Program`.

When the cursor is on the correct item, press either F1 or Ctrl-G to select that item. The submenu you selected then appears. To return to your previous menu, press either F4 or Ctrl-R. To go back to the Top Menu, press Ctrl-T.

ASSIST Commands

Notice the message `HELP: ^A` on the first line of the Top Menu. This message appears on all menus and command forms. When you type either Ctrl-A or F8, a list of ASSIST commands that can be used on that menu or command form appear on the screen. You can issue most of these commands by typing either a function key or a control character. (To remove the list of commands from your screen, press F2 or Ctrl-V or type any ASSIST command.)

The ASSIST commands are as follows:

Ctrl-A or F8	Lists ASSIST commands available on that menu or command form.
Ctrl-B	Moves the cursor back to the previous word on a command form.
Ctrl-C	Used in command forms to indicate characters that are ASSIST commands. For example, if you type Ctrl-B on a command form, you must precede it with Ctrl-C.

Ctrl-D	Exits the ASSIST program and returns you to the shell.
Ctrl-E	Allows you to escape ASSIST and execute one shell command. When you press Ctrl-E, the prompt `TYPE UNIX COMMAND:` appears. Type any shell command line or press Ctrl-V or F3 to kill the prompt. You will return to ASSIST after the shell command executes.
Ctrl-F or F5	Invokes the Pop-Up Menu.
Ctrl-G or F1	Selects the current item on a menu. It also executes a command line on a command form.
Ctrl-K	Stores a command line that you have built on a command form in a temporary file called *commandname.assist*.
Ctrl-L or F3	Redraws the screen. Pop-up help and messages are not redrawn.
Ctrl-N	Moves the cursor to the next item. If the current item is the last item on a menu or command form, the cursor moves to the first item.
Ctrl-O or F6	Prints the help message for the current menu or command form.
Ctrl-P	Moves the cursor to the item just above the current item on a menu or command form. If the current item is the first item, the cursor moves to the last item.
Ctrl-R or F4	Returns to the previous menu or command form screen. If there is not a previous menu or screen (that is, the current screen is the point at which you entered ASSIST), you will be asked if you want to exit ASSIST.
Ctrl-T	Returns to the Top Menu.
Ctrl-U	Moves to the next page of the current menu or command form.

Ctrl-V or F2	Clears a pop-up help message, the Pop-Up Menu, or a prompt line.
Ctrl-W	Moves to the next word on a command form.
Ctrl-Y	Displays the pop-up help message for the current item on a menu or command form.

Menu Layout

All ASSIST menus have a header line, a list of items, and a message line. The following screen shows the Compare Files Menu.

```
CURRENT MENU: COMPARE FILES     PREVIOUS MENU OR FORM: TOP        HELP ^A
------------------------------------------------------------------------

                COMPARE FILES

  >cmp ..... Show byte and line numbers of differences
   comm .... Select/reject common lines
   diff .... Show all lines that differ

          MORE COMMANDS FOR COMPARING FILES

   bdiff ... Show differences between files too large for "diff"
   diff3 ... Compare three files
   diffmk .. Create new file with "nroff" commands to mark differences
   sdiff ... Show files side-by-side, marking differences
```

The header line includes the names of both the current menu and the previous menu. On this screen, the header line tells you that the current menu is the Compare Files Menu and that you accessed this menu from the Top Menu. The header line also tells you that help is available.

The remainder of the menu lists the items you can select. The items can be either a list of other menus or a list of command forms. The current item is indicated by a greater-than sign (>) at the beginning of the line. If your terminal has highlighting, the item is also highlighted.

The Compare Files Menu lists the names of command forms. The list is divided into two groups. The first group (COMPARE FILES) lists the commands you can execute through ASSIST. The second group (MORE COMMANDS FOR COMPARING FILES) contains other commands associated with the menu category. Although you cannot execute these commands through ASSIST, you can see a brief description for any of these commands by moving to the item and then pressing Ctrl-Y.

The message line is at the bottom of the menu. A message appears here if you use a command inappropriately or choose a command form in the MORE COMMANDS FOR *menu name* section. There is no message line shown in this example.

Displaying Pop-Up Help

The ASSIST menus contain two types of pop-up help: menu help and item help. The menu help describes the menu itself; the item help describes a particular item on the menu.

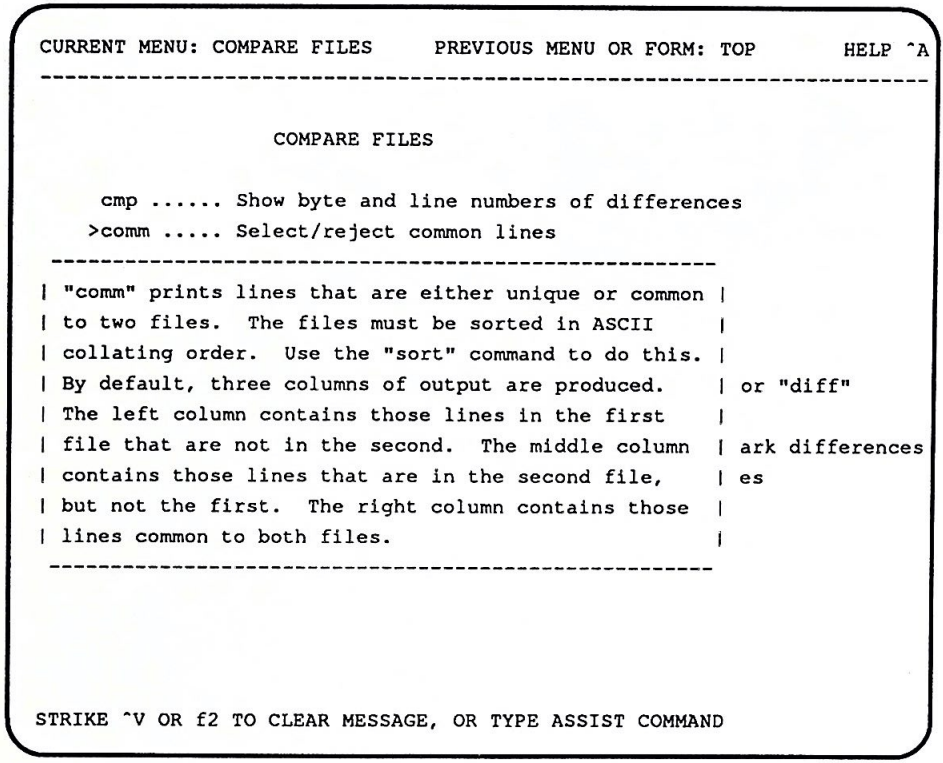
Menu Help. The menu pop-up help describes the current menu and gives you some examples of the commands it includes. To see the menu pop-up help, press either Ctrl-O or F7. For example, if you are on the Compare Files Menu, the pop-up help message looks like this:

```
CURRENT MENU: COMPARE FILES     PREVIOUS MENU OR FORM: TOP        HELP ^A
------------------------------------------------------------------------------
  ------------------------------------------------
 | The Compare Files menu lists commands that compare |
 | files and show the differences between files.      |
  ------------------------------------------------
     comm .... Select/reject common lines
     diff .... Show all lines that differ

            MORE COMMANDS FOR COMPARING FILES

     bdiff ... Show differences between files too large for "diff"
     diff3 ... Compare three files
     diffmk .. Create new file with "nroff" commands to mark differences
     sdiff ... Show files side-by-side, marking differences




 STRIKE ^V OR f2 TO CLEAR MESSAGE, OR TYPE ASSIST COMMAND
```

To remove the help message, press F2 or Ctrl-V or type any ASSIST command.

Item Help. The item pop-up help describes the current item (indicated by the greater-than sign (>)). If the current item is the name of an operating system command, the help message describes the function of the command. If the current item is the name of a submenu, the help message describes the the submenu and the commands it includes.

To see item pop-up help, press either Ctrl-Y or F6. The following screen shows the pop-up help for the **comm** command on the Compare Files Menu. To remove the help message, press F2 or Ctrl-V or type any ASSIST command.



5.4.3 Command Forms

The ASSIST command forms help you execute operating system commands. They describe the function of the command, its options, and its arguments. Command forms also help you understand the proper command line syntax by generating command lines for you. You can store the command lines you create and then reexecute them later.

Accessing Command Forms

To display the command form for a particular command, locate that command within the menu structure. Starting at the Top Menu, you can select the appropriate submenu, then select the command form from that submenu.

You can also bypass the menus and go directly to the command form. To do this, type **assist** followed by the name of the command. For example, if you want to use the **ls** command form, you would type this command:

```
$ assist ls Return
```

This command takes you directly to the **ls** command form.

Command Form Layout

Each ASSIST command form includes a header line, fields for specifying options and arguments, a command line, and an error message line.

If the command form has more than one page, ASSIST tells you the page you are on and the total number of pages in the command form. (This information appears near the bottom of the right side of the form.) Press Ctrl-U when you want to go to the next page.

The following pages show a filled-out **ls** command form. This form is two pages long and contains the following information:

Header Line

The header line tells you the name of the command form and specifies your location before you selected this form. In the example, the current form is `ls` and it was accessed directly from the `ls` Menu.

Option and Argument Specification Fields

These fields make up the body of a command form and describe the arguments and options that can be used with the command. A field consists of a description, the name of the option in brackets (when applicable), and sometimes a `(y/n)` to indicate that the field requires a yes/no answer.

On the **ls** command form, the first field requires that you enter the name of a file or directory. Most of the other fields require a yes or no answer.

On long forms, similar options might be grouped together. For example, the **ls** command form contains three groups of fields; `ATTRIBUTE LIST`, `SORTING OPTIONS`, and `OUTPUT FORMAT`.

```
CURRENT FORM: ls   PREVIOUS MENU OR FORM: ls menu                HELP: ^A
------------------------------------------------------------------------

      NAME OF INPUT FILE(S)/DIRECTORY(S):
      LIST NAMES BEGINNING WITH "." [-a] (y/n): y
      RECURSIVELY LIST SUBDIRECTORIES ENCOUNTERED [-R] (y/n): y
      LIST DIRECTORY NAME ONLY, NOT ITS CONTENTS [-d] (y/n): n


   ATTRIBUTE LIST
      LIST IN LONG FORMAT [-l] (y/n): y
      LIST IN LONG FORMAT, PRINT UID AND GID NUMBERS [-n] (y/n): n
      LIST IN LONG FORMAT, BUT DO NOT PRINT GROUP [-o] (y/n): n
      LIST IN LONG FORMAT, BUT DO NOT PRINT OWNER [-g] (y/n): n

   SORTING OPTIONS
      REVERSE ALPHABETIC OR OLDEST FIRST [-r] (y/n): y
      TIME MODIFIED [-t] (y/n): y
      TIME OF LAST ACCESS [-u] (y/n): n
      TIME OF LAST i-NODE MODIFICATION [-c] (y/n): n

------------------------------------------------------------PAGE 1 OF 2
 COMMAND LINE: ls -aRlrt > temp
------------------------------------------------------------------------
```

The last field on many command forms is labeled ADDITIONS TO THE COMMAND LINE. Enter any additional data here, such as piping or redirection commands. ASSIST does not check your entry in this field, so use it with care.

Command Line

The command line is the next-to-the-last line on the screen. It contains the operating system command line created from the information you type on the screen. When you add a piece of data to the command form, the command line is updated to reflect the new information.

```
CURRENT FORM: ls      PREVIOUS MENU OR FORM: ls menu              HELP ^A
------------------------------------------------------------------------

 OUTPUT FORMAT
    MULTI-COLUMN WITH ENTRIES SORTED VERTICALLY [[-C] (y/n): n
    MULTI-COLUMN WITH ENTRIES SORTED HORIZONTALLY [-x] (y/n): n
    STREAM OUTPUT [-m] (y/n): n

    APPEND "/" FOR DIRECTORIES, "*" FOR EXECUTABLE FILES [-F] (y/n): n
    APPEND "/" FOR DIRECTORIES [-p] (y/n): n
    PRINT NON-GRAPHIC CHARACTERS IN OCTAL/DDD NOTATION [-b] (y/n): n
    PRINT NON-GRAPHIC FILE NAME CHARACTERS AS "?" [-q] (y/n): n
    PRINT i-NUMBER OF FILE IN FIRST COLUMN [-i] (y/n): n
    GIVE SIZE IN BLOCKS FOR EACH ENTRY [-s] (y/n): n
    TREAT EACH ARGUMENT AS A DIRECTORY [-f] (y/n): n

    ADDITIONS TO THE COMMAND LINE: > temp


------------------------------------------------------------PAGE 2 OF 2
 COMMAND LINE: ls -aRlrt > temp
------------------------------------------------------------------------
```

Error Message Line

When you make an incorrect entry on a command form, the error message line prints either a left-adjusted error message or a right-adjusted warning message. Error messages appear for entries that will cause command execution to fail. Warning messages tell you of potential trouble. For example, you could receive a warning that you will overwrite an existing file. You can decide if you want to ignore the warning. Appendix A describes the ASSIST error messages.

Displaying Pop-Up Help

There are two types of help messages available on command forms: form help and field help. Form help messages describe the current form. Item help messages describe the field pointed to by the greater-than sign (>).

Form Help. The form help describes the function of the command. To display a form help message, press either Ctrl-O or F7. For example, this help message appears on the **ls** command form.

```
CURRENT FORM: ls   PREVIOUS MENU OR FORM: ls menu                HELP: ^A
------------------------------------------------------------------------
------------------------------------------------------------------------
'ls' lists contents of a directory or directories, and can give detailed
information about each listed file. This is the complete command form
which presents all of the 'ls' options. There is also a typical 'ls'
command form which provides you with choices for information about the
files. Choose an option from the SORTING OPTIONS group if you want to
order the list chronologically or reversed alphabetically. Choose an
option from the OUTPUT FORMAT group if you want to arrange your output
in columns or in stream format. To save the output in a file, use the
ADDITIONS TO THE COMMAND LINE field to redirect the output to a file.
------------------------------------------------------------------------

   SORTING OPTIONS
      REVERSE ALPHABETIC OR OLDEST FIRST [-r] (y/n): y
      TIME MODIFIED [-t] (y/n): y
      TIME OF LAST ACCESS [-u] (y/n): n
      TIME OF LAST i-NODE MODIFICATION [-c] (y/n): n

------------------------------------------------------------PAGE 1 OF 2
 COMMAND LINE: ls -aRlrt > temp
------------------------------------------------------------------------
```

To remove the help message, press F2 or Ctrl-V or type any ASSIST command.

Field Help. The help message for the current field describes the function of a command option or argument. It also tells you if any options are incompatible with the current item. To display pop-up help for the current field, press either Ctrl-Y or F6.

The next screen shows the pop-up help for the first field on the **ls** command form.

```
CURRENT FORM: ls   PREVIOUS MENU OR FORM: ls menu                HELP: ^A
------------------------------------------------------------------------

     NAME OF INPUT FILE(S)/DIRECTORY(S):
 ----------------------------------------------------------------------
 | List any number of file or directory names here.                   |
 | If you leave this blank, the current directory is listed.          |
 ----------------------------------------------------------------------

   ATTRIBUTE LIST
      LIST IN LONG FORMAT [-l] (y/n): y
      LIST IN LONG FORMAT, PRINT UID AND GID NUMBERS [-n] (y/n): n
      LIST IN LONG FORMAT, BUT DO NOT PRINT GROUP [-o] (y/n): n
      LIST IN LONG FORMAT, BUT DO NOT PRINT OWNER [-g] (y/n): n

   SORTING OPTIONS
      REVERSE ALPHABETIC OR OLDEST FIRST [-r] (y/n): y
      TIME MODIFIED [-t] (y/n): y
      TIME OF LAST ACCESS [-u] (y/n): n
      TIME OF LAST i-NODE MODIFICATION [-c] (y/n): n

------------------------------------------------------------PAGE 1 OF 2
 COMMAND LINE: ls -aRlrt > temp
------------------------------------------------------------------------
```

To remove the help message, press F2 or Ctrl-V or type any ASSIST command.

Moving the Cursor

The following keys move the cursor on a command form:

[Return] or [DownArrow]	Moves the cursor to the next input field
[Ctrl-P] or [UpArrow]	Moves the cursor to the input field just above the current field
[Ctrl-B] or [LeftArrow]	Moves the cursor to the left
[Ctrl-W] or [RightArrow]	Moves the cursor to the right
[Spacebar]	Edits a field and indicates the termination of a word. If a field already contains an entry, [Spacebar] erases the entry.

If you are on the last field on a form and press either [Return] or [DownArrow], the cursor moves to the first field on the screen. This "wrap" feature also applies to upward movement. If you are on the first field of the screen and press either [UpArrow] or [Ctrl-P], the cursor moves to the last field on the screen.

Some fields allow multiple entries, which must be separated by spaces. When you move the cursor to the left or right, you can move between the entries you have made on a line, but you cannot move from letter to letter in a particular entry. For example, if you type **file1 file2**, you can move from the word **file1** to the word **file2**, but you cannot move to the individual letters in the words **file1** and **file2**.

Entering Input

You can enter a single character, a single word, or multiple words in a field. Press [Return] after typing your entry. All printable characters appear in the input area.

The ASSIST command [Ctrl-C] is used on command forms to escape characters that have special meaning in ASSIST. (ASSIST commands such as [Ctrl-B] and shell metacharacters such as >, <, |, *, and " are examples of special characters.) For example, if you want to type [Ctrl-B] on a command form, you must first press [Ctrl-C], then press [Ctrl-B].

ASSIST provides shell expansion for characters that perform functions but are not printable themselves. When you enter one of these characters, they are represented on the command line by their ASCII equivalent. For example, Tab is shown as TAB.

Editing a Field

If you need to edit a field, use either UpArrow (or Ctrl-P) or DownArrow (or Return) to move the cursor to that field. If the field already contains input, type any printable character to delete the entry from that field. The character you typed then appears in the input area. Pressing Spacebar has the same effect; one word is removed each time you press Spacebar. Spacebar also indicates the end of a word.

For some fields, ASSIST supplies a value automatically. This is the default value and is in effect unless you change it. When you press Spacebar, ASSIST restores the default value, if there is one. If there is no default value, the previous input for this field is erased and the field is left blank.

To erase characters, press Backspace for each character that you want to erase.

Validating Data

ASSIST validates the command line whenever you perform these actions:

- Attempt to leave the current input area
- Press Spacebar (unless the function of this key is turned off by using quotation marks or backslashes)
- Try to execute the command line
- Store the command line

When ASSIST is validating the command line, it checks for errors that would result in an operating system error message. This includes checking for the existence of files, read permission, and command line syntax.

Here's a word of caution about validating fields. If a field contains several incorrect entries, you will receive an error message. However, if you edit that field and do not change all of the incorrect entries, ASSIST does not reissue the error message. Instead, it assumes that all corrections have been made.

When ASSIST is validating, a message such as the following might flash on the lower right corner of your screen.

```
VALIDATING INPUT, PLEASE WAIT ...
```

This message occurs when there is a time-consuming validation or when you leave a field.

The command line executes if all required arguments are present. ASSIST will not execute the command if required information is missing or has been specified incorrectly.

Executing a Command Line

After you have filled out a command form, you can press either F1 or Ctrl-G to execute the command. When you do this, the validation message might flash on your screen.

Some ASSIST command forms have execute messages that tell you about any special actions you must take to execute the command. Typically, these messages appear when you are about to run an operating system command that requires input from your terminal. If there is an execute message, it appears after ASSIST validates the command line. Otherwise, the command form disappears from the screen and a message that the command is executing appears.

After ASSIST executes the command, you can either press Return to go back to the same command form or press F4 or Ctrl-R to go to the previous screen. If you entered the command form directly from the operating system, pressing F4 or Ctrl-R returns you to the operating system prompt.

Saving a Command Line

ASSIST allows you to save any of the command lines you create on a command form. This feature is useful if you have to use a complex command frequently. For example, you might have created this command line on the **sort** command form:

```
COMMAND LINE: sort -rb +1.0 -1.1 infile1 infile2
```

To save this command line, press Ctrl-K. This message then appears on the bottom of your screen:

```
COMMAND LINE WRITTEN IN FILE sort.assist
```

ASSIST saves the command line in the file *sort.assist* in your current directory. It also assigns execute permission to the file. The *sort.assist* file will be overwritten the next time you store a new command line for **sort** in this directory. If you want to save this file, you should give it a different name (use the **cp** or **mv** command). When you want to reexecute the saved command line, type the name of the file where the line is stored. In this case, you would type **sort.assist**.

5.4.4 The ASSIST Walkthrus

The ASSIST walkthrus are interactive simulations of operating system commands or concepts. Walkthrus allow you to begin using the command or concept immediately. ASSIST includes walkthrus for ASSIST, the operating system, and the **vi** screen editor.

You can access the ASSIST walkthru and the operating system walkthru from either the Pop-Up Menu or the System, Terminal, and General Information Menu. The vi walkthru can be accessed from the Edit, Cut, and Paste Files Menu. You can also access the walkthrus by typing one of these commands:

- $ **assist assistwalk** [Return]
- $ **assist unixwalk** [Return]
- $ **assist viwalk** [Return]

When you are in a walkthru, you can type [Ctrl-A] or [F8] to display a list of ASSIST commands that apply to walkthrus. To use an ASSIST command in a walkthru, you must first press either [Ctrl-A] or [F8]. Then type the ASSIST command.

To exit a walkthru, first press [Ctrl-A] or [F8], then press one of the following keys:

- [Ctrl-R] or [F4] to return to the previous menu
- [Ctrl-T] to return to the Top Menu
- [Ctrl-D] to exit **assist**

5.4.5 The Pop-Up Menu

The Pop-Up Menu gives you access to command forms, the ASSIST walkthru, and the operating system walkthru. It also includes a command search feature and allows you to change directories, return to the shell temporarily, or exit ASSIST. The Pop-Up Menu, like the full-screen menus, also includes a help message.

To access the Pop-Up Menu, type either F5 or Ctrl-F while you are on any ASSIST menu or command form. If you selected the Pop-Up Menu and then decide you don't need to use it, press either Ctrl-R or F4 to return to your previous menu or command form.

The following screen shows the Pop-Up Menu overlaid on the Compare Files Menu.

```
CURRENT MENU: COMPARE FILES     PREVIOUS MENU OR FORM: TOP        HELP ^A
------------------------------------------------------------------------------

 ---------------------
| Assist Walkthru     |RE FILES
| Command Forms       |
| Command Search      | and line numbers of differences
| Exit ASSIST         |eject common lines
| Switch Directory    | lines that differ
| UNIX System Subshell|
| UNIX System Walkthru|S FOR COMPARING FILES
 ---------------------
   bdiff .... Show differences between files too large for "diff"
   diff3 .... Compare three files
   diffmk ... Create new file with "nroff" commands to mark differences
   sdiff .... Show files side-by-side, marking differences
```

To select an item from the Pop-Up Menu, move the cursor to that item, then press either F1 or Ctrl-G.

The Pop-Up Menu contains the following items:

ASSIST Walkthru

The `ASSIST Walkthru` offers brief, interactive demonstrations of ASSIST concepts and commands. When you select this item, a menu appears. You can choose the complete walkthru, which covers all of ASSIST, or you can choose a shorter walkthru on a particular aspect of ASSIST.

Command Forms

The Pop-Up Menu eliminates the need to use the full-screen menus to access command forms. When you select `Command Forms`, a prompt appears on the bottom of the screen. You can either type the name of a command or select the Command Forms Menu, which lists all available command forms.

If you know which command form you want to use, type the command name. For example, to display the **rmdir** command form, type **rmdir**. If you name a command form that does not exist, a message stating that ASSIST does not support that command will appear at the bottom of the screen. When you press Return, the command form prompt reappears on the screen.

If you want to see the Command Forms Menu, simply press Return at the command form prompt. This menu lists the commands supported by ASSIST and gives a brief description of their functions. To exit the Command Forms Menu, press either Ctrl-R or F4. You will then return to your previous menu or form.

The `Command Search` feature helps you find the name of a command. When you select this item, the following prompt appears on the bottom of the screen:

```
TYPE SEARCH WORD:
```

The *search-word* you enter should be as unique to the command as possible. For example, if you want to remove a file, you could use either **remove** or **file** as the *search-word*. If you use **file** as the *search-word*, you will get a very long list of commands.

When you enter a *search-word* and press Return, ASSIST scans a database containing descriptions of the commands in the *Reference Manual*. ASSIST then lists all commands having the *search-word* in their description. For example, if the search word is **remove**, all commands containing **remove** in their description are listed. If there

is more than one page of output from the search, press Return to continue the search.

After all of the appropriate commands have been listed, this prompt appears at the bottom of your screen:

```
STRIKE RETURN TO CONTINUE WITH MENU, OR TYPE COMMAND NAME:
```

If you find the command you want to use, type the name of the command, press Return, and the appropriate command form will appear on the screen.

If you want to go back to the previous menu, simply press Return.

Exit ASSIST

Select this item if you want to leave the ASSIST program. It has the same effect as typing Ctrl-D; it returns you to the operating system command level.

Switch Directory

This item allows you to change directories while you are working in ASSIST. You will remain in the directory you specify until you either exit ASSIST or select this item again. When you choose `Switch Directory`, the following prompt appears at the bottom of the screen:

```
CURRENT DIRECTORY: <current directory name> NEW DIRECTORY:
```

This prompt tells you the name of your current working directory and asks for the name of the directory you want to move to. (If your current directory has a long name, you will only see the `NEW DIRECTORY:` prompt on the screen.)

At this prompt, type the directory you want to move to. If you specify a valid directory, a message appears in this form:

```
MOVED TO: <new directory name> [TYPE RETURN]:
```

If the directory name is invalid, this error message appears at the bottom of the screen:

```
DIRECTORY DOES NOT EXIST OR IS NOT ACCESSIBLE [TYPE RETURN]:
```

At this point, you are prompted again for a new directory name.

UNIX System Subshell

The `UNIX System Subshell` takes you out of ASSIST temporarily and allows you to enter operating system commands. When you select this item, the screen clears and the prompt `<ASSIST>`

appears. You can enter operating system commands at this prompt. When you finish your operating system tasks, press Ctrl-D to return to ASSIST. You will return to the location you were in when you left ASSIST.

UNIX System Walkthru

The `UNIX System Walkthru` provides brief, interactive demonstrations of operating system concepts and commands. When you choose this item, a menu appears listing the items such as files, directories, input, output, and redirection that are available in the walkthru.

Chapter 6
Using the Printer

6.1 Introduction 6-1

6.2 The LP Print Service 6-1

6.3 Printing a File: lp 6-2
lp Options 6-3
Specify Another Printer 6-3
Copy the File Before Printing 6-3
Print Specific Pages from a File 6-3
Print Multiple Copies of a File 6-4
Send Notification when Job Is Printed 6-4
Cancel the Banner Page 6-5
Print Files Continuously 6-5
Specify Page Size and Pitch Settings 6-6
Change Queue Priority 6-7
Print a Form 6-7
Specify Character Sets, Cartridge Fonts, or Printwheels 6-8
Specify Special Printing Modes 6-8

6.4 Printing Command Output 6-9

6.5 Displaying LP Status: lpstat 6-10
Display the Default Destination 6-10
Display Print Requests for a User 6-11
Display Print Requests for a Printer 6-11
Display Printer Status 6-11
Display Forms Information 6-12
Display Character Set and Printwheel Information 6-12

6.6 Changing a Print Request 6-13

6.7 Canceling a Print Request: cancel 6-14

6.8 Enabling and Disabling Printers: enable, disable 6-15

Chapter 6
Using the Printer

6.1 Introduction

Some terminals have built-in printers that print paper copies of files. If you have this type of terminal, you can print a file simply by turning on the printer and entering the **cat** or **pr** command. However, if you are using a video display terminal, you must generate a printing request. These requests go through the LP print service, which stores the requests on disk and then sends them to the appropriate printer.

This chapter tells how to use the LP print service to generate or cancel print requests, to check the status of printers and print requests, and to enable or disable a printer.

6.2 The LP Print Service

The LP print service allows you to send print jobs to a printer. The system spools (or stores) your print requests on disk, allowing you to perform other tasks while your job is printing. When there are multiple requests for a single printer, the print requests are placed in a queue, and the printer prints each job in turn.

When the LP print service was set up at your site, your system administrator specified one printer as the default destination for all print requests. You can send your print requests to this printer, or you can specify another printer when you create a print request. You can also assign your own default destination printer. To do this, set the **LPDEST** environment variable, as described in Chapter 4.

If your site has several printers, your system administrator can set up classes of printers. You can then send a print request to a particular class, and the first available printer in that class will print it.

When you are printing a file, be sure you don't remove the file or make any changes to it until the print job is complete. If you change the file before it is printed, the job might not print properly. If you will need to change a file before it is printed, use the **–c** option, which tells the system to copy the file before printing it. (This option is described later in this chapter.)

6.3 Printing a File: lp

The **lp** command (short for *line printer*) allows you to generate print requests. You can print files on your system's default destination printer or on another printer that you specify.

To generate a print request for the default printer, type **lp** followed by the name of the file you want to print. For example, to print the *memos* file, you would type this command:

```
$ lp memos Return
```

The system will then display both the name (or class) of the printer that will print the file and the identification number (called a request ID) assigned to the request. In this example, the printer named `laser1` will print the *memos* file and the request ID is `laser1-6885`.

```
$ lp memos Return
request id is laser1-6885  (1 file)
$
```

You can specify more than one file on the command line. For example, to print the *memos* and *notices* files, you would type the command shown on the following screen. The system responds with the message: `(2 files)`.

```
$ lp memos notices Return
request id is laser1-7224  (2 files)
$
```

Other Information

lp(1)

6.3.1 lp Options

The following sections describe several options for the **lp** command. The use of some of these options is dependent on the types of printers available at your site and the way the LP print service is set up. Check with your system administrator to determine the options you can use.

Specify Another Printer

When you issue the **lp** command as shown in the previous section, your file prints on the default destination printer. If your system has several printers, you can specify that the file print on another printer or class of printers. (Your system administrator can tell you the printers or classes of printers available to you.)

To specify another printer or class of printers, use the **–d***destination* option when you issue the **lp** command. For example, to send the *memos* file to the *laser2* printer instead of the default printer, you would type this command:

```
$ lp -dlaser2 memos  Return
request id is laser2-7834    (1 file)
$
```

Copy the File Before Printing

The **–c** option causes the system to make a copy of the file at the time it receives the print request. The system then uses the copy to print the file. If you don't use this option, be sure you don't remove the file or make any changes to it until the print job is complete. If you change the file before it is printed, the job might not print properly.

Print Specific Pages from a File

The **–P** option allows you to print specific pages or a series of pages. (This option might not be available at your site. Check with your system administrator to determine if you can use this option.)

To print specific pages from a file, type **–P**, then type the numbers of the pages you want to print. Use commas to separate the page numbers. To indicate a series of pages, type the first page number in the series, followed by a hyphen. Then type the last page number. The following command prints pages 5 through 9, 27, and 29 from the *chapter4* file.

```
$ lp –P5-9,27,29 chapter4 Return
```

You can list the page numbers in any order; the system will print the pages in numeric order. It will also drop any duplicate page requests.

Print Multiple Copies of a File

The **–n***number* option lets you specify the number of copies to print. To use this option, type **–n** followed by the number of copies. For example, the following command specifies that **lp** print three copies of the *johnson.memo* file. The command also includes the **–c** option.

```
$ lp -c -n3 johnson.memo Return
request id is laser1-7224   (1 file)
$
```

Send Notification when Job Is Printed

The LP spooler system can notify you when your print job is complete. If you want the system to send you mail, type **–m** on the command line, as shown on the following screen.

```
$ lp -m johnson.memo Return
request id is laser1-7352   (1 file)
$
```

If you want the system to display a message on your screen, type **-w** on the command line.

```
$ lp -w johnson.memo  Return
request id is laser1-7412    (1 file)
$
```

If you are logged out when the job prints, the system will send you mail instead of displaying a message.

Cancel the Banner Page

The print service prints a banner (or title) page for each print request. The banner pages make it easy to separate print jobs and to identify the owner of each job. If you don't want a banner page to print for a particular job, use the **-o nobanner** option. (Your system administrator can turn this option off for specific printers.)

```
$ lp -o nobanner joe.letter  Return
request id is laser1-7553    (1 file)
$
```

Print Files Continuously

When you specify several files in a single print command, the system advances to a new page when it starts printing a new file. If you don't want each file to start on a new page, use the **-o nofilebreak** option.

```
$ lp -o nofilebreak job_a job_b job_c  Return
request id is laser1-7623     (3 files)
$
```

Specify Page Size and Pitch Settings

When adding a printer to the LP service, the system administrator can set default values for the page size (length and width) and pitch settings (lines per inch and characters per inch). These settings are dependent on the type of printer. For example, some printers can accommodate wide paper; other printers are limited to 8-1/2 inch paper.

If you want to use a different page size or pitch setting for a particular print job, you can include the settings on the **lp** command line. Type the **–o** option followed by your specifications. (Include only the specifications you want to change from the default values.) Your command should have this form:

$ **lp –o "length=***sdn* **width=***sdn* **lpi=***sdn* **cpi=***sdn***"** *filename*

lpi represents lines per inch or centimeter; **cpi** represents characters per inch or centimeter. *sdn* represents a scaled decimal number. This is a non-negative number that indicates a unit of size. Following are the options for each specification:

Page length

- Type a number to indicate the lines per page.
- Type a number followed by **i** to indicate inches per page.
- Type a number followed by **c** to indicate centimeters per page.

For example, **length=66** indicates a page length of 66 lines, **length=11i** indicates a page length of 11 inches, and **length=27.94c** indicates a page length of 27.94 centimeters.

Page width

- Type a number to indicate the columns per page.
- Type a number followed by **i** to indicate the width in inches.
- Type a number followed by **c** to indicate the width in centimeters.

Lines per inch

- Type a number to indicate the number of lines to print per inch.
- Type a number followed by **c** to indicate the number of lines to print per centimeter.

Characters per inch

- Type a number to indicate how many characters to print per inch.
- Type a number followed by **c** to indicate the number of characters to print per centimeter.
- Type **pica** for 10 characters per inch.
- Type **elite** for 12 characters per inch.
- Type **compressed** for the maximum number of characters the printer can print per inch.

When you change page size or pitch specifications, keep in mind that some of these options might not apply to your printer. You cannot change page or pitch settings if the command line includes the **–f** option to print a form.

Change Queue Priority

When you issue a print request, your print job is placed in a print queue for the appropriate printer. Print requests are also assigned a priority level that determines their placement in the queue. Priority levels range from 0 (zero), the highest priority, to 39. The default priority level is typically 20; your system administrator can change this to another number.

If you want to change the priority level for a particular print request, include the **–q** option, followed by the priority level, on the command line. The following command assigns priority 0 to the *important* file. This level sends the file to the front of the print queue.

```
$ lp –q 0 important Return
```

The system administrator can limit the priority levels you can use. If you specify a priority level that is disallowed to you, your request will print with the default priority level. Your system administrator can tell you the default priority level for your site and the priority levels you can use.

Print a Form

The LP service can manage the printing of preprinted forms such as checks or company letterhead. If your site is using this option, your system administrator will assign a name to each type of form you are using and will specify the printers that can print each type of form. (You can use the **lpstat** command to determine the name assigned to a form.)

When you want to print a form, include the **–f** option, followed by the name of the form, on the command line. The **–d any** option tells the LP service to send the print request to any printer that can handle that type of form.

$ **lp –d any –f invoice may.inv** Return

The LP service will notify your system administrator that you need to print a form (**invoice** in this example). The system administrator can then load the appropriate form and print your job.

You can also use the **–d** option to send the print request to a particular printer. However, if that printer cannot print the form you specified, the system will reject your print request.

Specify Character Sets, Cartridge Fonts, or Printwheels

The system administrator can also assign names to the character sets, cartridge fonts, or printwheels available on each printer. If your site is using this option, the system administrator will be notified when you issue a print request that requires a different cartridge font or printwheel. The system administrator can then mount the cartridge or printwheel and print the job. A change in character set does not require intervention from the system administrator.

If your print job requires a particular character set, cartridge font, or printwheel, include the **–S** option, followed by the appropriate name, on the command line.

$ **lp –S Courier table** Return

This example specifies a Courier printwheel. Because printwheels (and cartridge fonts) must be mounted on the printer, this job will remain in the queue until the system administrator mounts the printwheel and then tells the LP service to print the job.

Specify Special Printing Modes

Your system administrator can set up special printing modes for your site. For example, you might want your text to print sideways on the page (landscape mode), or you might want to print on both sides of the page. The special modes available to you are dependent on the types of printers you have.

A name is assigned to each special mode. To use these modes, type the **–y** option followed by the name of the mode. The following command prints the *chart* file in landscape mode.

$ **lp –y landscape chart** Return

6.4 Printing Command Output

You can use the pipe (|) output redirection symbol described in Chapter 4 to redirect the output from a command to your printer. For example, the following command line prints the directory listing produced by the **ls –l** command.

```
$ ls -l | lp  Return
request id is laser1-8230    (1 file)
$
```

You can also use this method to print paginated files. For example, you can issue the **pr** command to paginate a file and then use the pipe to send the output from **pr** to the printer.

```
$ pr trip.report | lp  Return
request id is laser1-8343    (1 file)
$
```

You can follow the **lp** command with any of its options. The following command line includes the **–n** option to print two copies of the output from the **cat** command.

```
$ cat agenda |  lp -n2  Return
request id is laser1-8456    (1 file)
$
```

6.5 Displaying LP Status: lpstat

The **lpstat** command lists status information for printers and print requests. You can use this command to determine a request ID number, to identify the default destination printer for your system, to display all print requests for a specific user, to look at the queue for a particular printer, and to determine if a printer is active.

To determine the status of your own print requests, execute the **lpstat** command, as shown on the following screen. The system then lists the status of your print requests, including the request ID, the size of the file in bytes, and the time you made the request.

```
$ lpstat  Return
laser1-6885     jimh     173472     May 12 18:02
laser2-8434     jimh     294853     May 12 18:03
$
```

The following sections describe some **lpstat** options you might find useful.

Display the Default Destination

The **–d** option tells you which printer is the default destination for your system. If you do not specify a printer or class of printers on the command line, your print requests will go to this printer. If the default printer is *laser1*, the following message will appear in response to the **–d** option:

```
$ lpstat -d  Return
system default destination: laser1
$
```

Display Print Requests for a User

The **-u** option prints the status of print requests made by specific users. To obtain status information for a particular user, follow **-u** with the user's login name, as shown in the following example.

```
$ lpstat -u mary  Return
laser1-6892     mary     100124     May 12 12:42
laser2-8381     mary     214268     May 12 13:03
$
```

Display Print Requests for a Printer

The **-o** option prints the status of print requests generated for a particular printer. To look at the print queue for a specific printer, type **-o** followed by the name of the printer.

```
$ lpstat -o laser1  Return
laser1-6885     starship     173472     May 12 18:02
laser1-6886     mary         194236     May 12 18:03
$
```

Display Printer Status

The **-p all** option prints the status of all printers. This option will tell you if a printer is enabled (active) or disabled (inactive). To print information for specific printers, follow **-p** with the names of the printers, as shown on the following screen. Use commas to separate the printer names.

```
$ lpstat -p lp1,lp2  Return
printer lp1 is now printing lp1-6885. enabled since May 12 18:02
printer lp2 is idle. enabled since May 4 12:45
$
```

Display Forms Information

The **–f** option prints information about the forms your system administrator has defined in the LP print service. To print information about all forms, type this command:

```
$ lpstat -f all  Return
form invoice is available to you, mounted on laser-1
form check is not available to you
$
```

The system administrator can restrict user access to any form. If you can print a form, the message `is available to you` follows the name of the form. If you cannot print a form, the message `is not available to you` appears. If the form is currently loaded on a printer, a message such as `mounted on laser-1` also appears.

For more detailed information about a form, include the **–l** option on the command line. The system will then print a description of the form, its dimensions, number of pages, and so on.

$ **lpstat –f all –l** Return

Display Character Set and Printwheel Information

The **–S** option prints a list of all character sets and printwheels available at your site. The **–l** option tells you the printers that can use each character set or printwheel.

$ **lpstat –S all –l** Return

Other Information

lpstat(1)

6.6 Changing a Print Request

The **–i** option to the **lp** command allows you to change a previous print request. If the request is currently printing, it will be stopped and then restarted with the changes you specify.

To change a print request, type **lp –i** followed by the request id the system assigned to the original request. Then type only those options you want to change. For example, your original command might look like this:

```
$ lp -dlaser2 -n2 jackson  Return
request id is laser2-6434    (1 file)
$
```

If you want to change the number of copies to be printed, you would type this command:

$ lp –i laser2-6434 –n3 Return

The previous example includes only the options that are to change in the revised print request. There is one exception to this rule. If your original print request includes **–o**, which can be followed by one or more options, the new command must include all **–o** options that are to apply to the revised print request. For example, your original command might include this **–o** option:

```
$ lp -n3 -o nobanner jackson schedule  Return
request id is laser1-7823    (2 files)
$
```

If you want to change the print request to print the files without a page break (**–o nofilebreak**), you would type this command:

$ lp –i laser1-7823 –o nobanner nofilebreak Return

Other Information

lp(1)

6.7 Canceling a Print Request: cancel

The LP spooling system allows you to cancel print requests. You can cancel either a specific print request or the print request that is currently printing. After the request is canceled, the printer goes on to the next request in its queue. You can cancel only your own print requests.

To cancel a specific print request, type **cancel** followed by the request ID number. (To determine a request ID number, use the **lpstat** command described earlier.) For example, to cancel your request to print the *memos* file (request ID `laser1-6885`), you would type this command:

```
$ cancel laser1-6885  Return
request "laser1-6885" canceled
$
```

This command cancels the print request even if the job is currently printing.

To cancel the current print request for a particular printer, type **cancel** followed by the name of the printer. For example, to cancel the job currently printing on the *laser2* printer, you would type this command:

```
$ cancel laser2  Return
request "laser2-4523" canceled
$
```

Other Information

lp(1) (includes **cancel**)

6.8 Enabling and Disabling Printers: enable, disable

When a printer can print the requests sent to it, it is said to be *enabled*. If a printer cannot print the requests, it is *disabled*. The **enable** command enables a printer; the **disable** command disables a printer.

Typically, printers are disabled for maintenance purposes or to perform tasks such as changing printwheels or loading forms. When a printer is disabled, the print service will continue to accept print requests for that printer. The requests will print when the printer is enabled.

To determine if a printer is currently enabled or disabled, use the **lpstat -p** command described earlier.

Some sites allow all users to enable and disable printers. At other sites, only the system administrator can perform these tasks.

To enable a printer, type **enable** followed by the printer name. The command on the following screen enables the *laser1* printer.

```
$ enable laser1  Return
printer "laser1" now enabled
$
```

To disable a printer, type **disable** followed by the printer name. For example, the following command disables the *laser2* printer.

```
$ disable laser2  Return
printer "laser2" now disabled
$
```

When you disable a printer, it stops printing the current print request. This request will be completely reprinted when the printer is enabled again. (If you have another printer in the same class, the request will be printed on that printer instead.)

If you want the current request to finish printing before the printer is disabled, add the **–W** option to the command line. This option tells the system to finish printing the current request, then disable the printer.

```
$ disable –W laser2 Return
```

If you want to cancel the current request and then disable the printer, add the **–c** option to the command line.

```
$ disable –c laser2 Return
```

The **–r** option allows you to include a reason for disabling the printer. To use this option, type **–r** followed by the reason, which must be enclosed in quotation marks. For example, if you were disabling the *laser2* printer for maintenance, you could make this entry on the command line:

```
$ disable -r "maintenance" laser2 Return
printer "laser2" now disabled
$
```

If you then used the **lpstat –p** command to check the status of this printer, the reason you specified would appear.

Other Information

enable(1) (includes **disable**)

Chapter 7
Managing Processes

7.1 Introduction 7-1

7.2 Creating Processes 7-1

7.3 Managing Processes in the Bourne Shell 7-2
Displaying the Status of Running Processes: **ps** 7-2
Terminating Active Processes: **kill** 7-3

7.4 Managing Processes in the Korn Shell or C Shell 7-4
Displaying the Status of Processes: **jobs** 7-4
Killing Processes: **kill** 7-5
Stopping and Resuming Processes 7-5

7.5 Running Commands at a Later Time: batch, at 7-7
Displaying **at** or **batch** Job Information 7-9
Canceling an **at** or **batch** Command 7-9

7.6 Running Processes after Logout: nohup 7-10

Chapter 7
Managing Processes

7.1 Introduction

When the operating system executes a command, the execution is called a *process*. It is important to distinguish between processes and programs. A program is an executable file; a process is the execution of the program. If several instances of a program are running at the same time, each instance of the program is a separate process.

This chapter tells how to manage processes. If you are using the Bourne shell, you can display or kill your processes. If you are using either the Korn shell or the C shell, you can display the status of your processes, suspend or kill them, and move them to the foreground or background. This chapter also tells how to run commands at a later time and how to run commands after you log out.

7.2 Creating Processes

When the operating system is started up, a process is created. This process is the parent of all processes created by users or the operating system. To keep track of these processes, the operating system assigns a process identification number (or PID) to each process.

When you log onto the system, the operating system creates a process to run your shell program. This process will run the shell throughout the login session and will keep the same PID number.

When you issue a command, the process running the shell *forks*, or creates, a child process to execute the command. While the child process runs the command, the parent process (which is running the shell) remains inactive. When the child process completes, the parent process is reactivated and prints the shell prompt.

If you run a command in the background, the shell forks a child process to execute the command. However, the shell does not wait for the child process to complete the command. Instead, it displays the PID number of the child process followed by the shell prompt, allowing you to enter another command.

7.3 Managing Processes in the Bourne Shell

If you are working in the Bourne shell, you can use the commands described in this section to display or kill processes.

7.3.1 Displaying the Status of Running Processes: ps

The **ps** command displays the status of the processes you are running in the background. In the following example, we ran **grep** in the background, then issued the **ps** command.

```
$ grep word * > temp &  Return
28223
$
$ ps  Return
PID    TTY          TIME COMMAND
28124  ttyGA/GABB 0:00   sh
28223  ttyGA/GABB 0:04   grep
28224  ttyGA/GABB 0:04   ps
$
```

The system responds with the process identification (PID) number assigned to each process. It also identifies the terminal you used (TTY), gives the cumulative execution time (TIME) for each process, and the prints the name of each command (COMMAND).

In our example, the output includes the **grep** command, the **ps** command itself, and the **sh** (*shell*) command that runs while you are logged in.

The options to the **ps** command provide additional information about each process. For example, the **ps –l** command displays a long listing that includes the PID number assigned to each parent process. The **–e** option displays information about all running processes, including those belonging to other users.

Other Information

ps(1)

7.3.2 Terminating Active Processes: kill

The **kill** command terminates processes running in the background. You can kill only your own processes.

To terminate a process, type **kill** followed by the PID number assigned to that process. (You can use the **ps** command to obtain the appropriate PID number.) On the following screen, we used **kill** to terminate the **grep** command shown in the previous example.

```
$ kill 28223 Return
28223 Terminated
$
```

The system responds with a termination message and the shell prompt ($) appears. If the system cannot find the PID number you specify, it responds with an error message.

```
kill:28228:No such process
```

The **kill** command shown on the previous screen might not terminate all processes. If a process is still running, you can issue **kill -9** followed by the PID number. This command always kills the process you specify.

Other Information

kill(1)

7.4 Managing Processes in the Korn Shell or C Shell

Both the Korn and C shells include a feature called *job control* that tracks the processes you are running. You can use job control to display the status of your processes, to kill or stop processes, and to move processes to the foreground or the background. Although the examples in this section show the Korn shell, the job control commands work the same way in the C shell.

The shell assigns a job number to each command you send to the background. The job number, enclosed in brackets, appears on the screen after you type the command. The process number (PID) also appears.

```
$ grep alpha chap? > temp &  Return
[1]     15323
$
```

When the command (or job) is complete, the shell prints a status message before the next shell prompt appears.

```
[1] +  Done            grep alpha chap? > temp &
$
```

Other Information

csh(1), **ksh**(1)

7.4.1 Displaying the Status of Processes: jobs

The **jobs** command prints the status of background jobs that have not been completed. The following screen shows sample output from this command.

```
$ jobs  Return
[3] +  Running          pr chap1 | lp &
[2] -  Running          grep machine spec* > machine &
[1]    Running          pr chap5 | lp &
```

The plus sign (+) indicates the current job (the last command you sent to the background). The minus sign (–) indicates the previous job.

The **jobs –l** command also prints the PID number.

7.4.2 Killing Processes: kill

The shell includes a command called **kill** that terminates jobs running in the background. To use this command, type **kill**, then type either the process number or a percent sign (%) followed by the job number. For example, to kill the **grep** command on the previous screen, you could type this command:

```
$ kill %2  Return
[2] -  Terminated        grep machine spec* > machine &
$
```

In place of the job number, you can enter a plus sign to kill the current job (**kill %+**) or a minus sign to kill the previous job (**kill %–**). You can also type the first characters of the command. For example, the following line would also kill the **grep** command:

```
$ kill %grep  Return
```

7.4.3 Stopping and Resuming Processes

If a process is running in the foreground and you want to suspend it, press Ctrl-Z. A message telling you the job is stopped then appears on the screen.

```
$  grep drawing spec* > drawings  Return
Ctrl-Z [1] +  Stopped      grep drawing spec* > drawings
$
```

When you want the job to resume, use either the **fg** or the **bg** command. The **fg** command continues the job in the foreground; the **bg** command continues the job in the background. For example, the following command continues the job stopped in the previous screen and places it in the background.

```
$ bg  Return
[1]          grep drawing spec* > drawings
```

The command on the previous screen restarted the current job. If you want to restart a different job, follow the **fg** or **bg** command with a percent sign (%). Then type either the job number, a plus sign (to restart the current job), or a minus sign (to restart the previous job). For example, if you want to restart job number 2 in the foreground, you would type **fg %2**.

If you try to leave the shell while jobs are running or stopped, a message such as `You have stopped jobs` will display. If you want to look at the jobs, use the **jobs** command. You can then restart or kill the jobs, or wait for them to finish processing. If you try to leave the shell again, you will be allowed to exit and a message will not appear. Any stopped jobs will be terminated.

7.5 Running Commands at a Later Time: batch, at

The **batch** and **at** commands allow you to specify a command or sequence of commands to run at a later time. If you use the **batch** command, the system determines when the commands will run. If you use the **at** command, you determine when the commands will run.

The **batch** command is useful if you are running a process or shell program that uses a large amount of system time. The **batch** command submits a batch job (containing the commands to be executed) to the system. The job is put in a queue and runs when the system load falls to an acceptable level. This frees the system to respond rapidly to other input and is a courtesy to other users.

The **batch** command has this format:

batch Return
first command Return
.
.
.
last command Return
Ctrl-D

Type the **batch** command, then type each command to be included in the batch job on a separate line. Press Ctrl-D when your list is complete. If you are executing a single command, you can type **batch** and the command on the same line.

The following example uses **batch** to execute a **grep** command. The system displays a job number and the date and time the command will execute. (The job number is not the same as the PID number the system generates when you issue a command.)

```
$ batch grep dollar * > dol.file  Return
Ctrl-D
job 155223141.b at Sun Dec 7 11:14:54 1986
$
```

The **at** command allows you to specify when the commands should execute. The **at** command has this format:

at *time* [*date*] [+ *increment*] Return
first command Return
.
.
.
last command Return
Ctrl-D

The *time* argument specifies the time of day the commands should execute. You can use a 1-, 2-, or 4-digit number to represent the time. 1- or 2-digit numbers specify hours; 4-digit numbers specify hours and minutes. You can use a colon within a 4-digit number to indicate *hour:minutes*. For example, you could type either 1030 or 10:30. You can follow the time with either **am** or **pm**, as in 10:30am. If you omit this suffix, the command assumes a 24-hour clock. You can also type **noon**, **midnight**, **now**, or **next** instead of a number.

If you want the commands to execute today, you can omit the *date* argument. If you want to execute the commands on a different date, type either the date (such as Feb 24 or May 18) or the day of the week. You can spell out the full name of the day or abbreviate it to three letters (Monday or Mon). You can also enter the words **today** or **tomorrow**.

The *increment* is a number followed by either **minutes**, **days**, **weeks**, **months**, or **years**. The **at** command adds the increment to the time and date you specified. This argument is optional.

Following are some valid **at** command lines:

```
$ at 0945am Jul 14
$ at 09:45am Jul 14
$ at now + 1 day
$ at 3 pm Thursday
```

After typing the **at** command line, list the commands to be included in the job. Type each command on a separate line and press Ctrl-D when you finish the list.

The following example uses the **at** command to mail the *birthday* file to *emily*. The system then prints the job number and the date and time the job is to execute.

```
$ at 8:15am Feb 27 Return
mail emily < birthday Return
Ctrl-D
job 453400603.a at Mon Feb 27 08:15:00 198
$
```

7.5.1 Displaying at or batch Job Information

The **–l** option to the **at** command displays a list of the current jobs in the **batch** or **at** queue. The output includes job numbers and the date and time each job will run.

```
$ at -l Return
user = jean 168302040.a at Sat Feb 25 13:00:00 1989
user = jean 453400603.a at Mon Feb 27 08:15:00 1989
$"
```

7.5.2 Canceling an at or batch Command

If you decide not to execute the commands currently waiting in a **batch** or **at** job queue, you can use the **–r** option of the **at** command, followed by the job number, to cancel the job.

For example, to cancel the last **at** job on the previous screen, you would enter this command:

at –r 453400603.a Return

If you don't know the job number, use the **at –l** command to display a list of current jobs and the associated job numbers.

You can cancel only your own jobs.

Other Information

at(1)

7.6 Running Processes after Logout: nohup

Normally, all processes are killed when you log off. If you want a background process to continue running after you log off, you must use **nohup** to issue the command. The **nohup** command has this format:

nohup *command* **&**

Type the **nohup** command, then type the command to run in the background. For example, you could issue the following **grep** command and then log off immediately. The **grep** command will continue to execute after you log off.

```
$ nohup grep word * > word.list & Return
```

If necessary, you can use the **kill** command to terminate the **nohup** command.

Other Information

nohup(1)

Chapter 8
The ed Line Editor

8.1 Introduction 8-1
Concepts 8-1
Operating Modes 8-1
The Buffer 8-1
Line Addresses 8-2
ed Commands 8-2
Sample ed Sessions 8-2
Creating a New File 8-3
Editing an Existing File 8-5

8.2 Invoking ed 8-8
Specifying an **ed** Prompt 8-9

8.3 Writing to a File and Quitting ed 8-9
Writing the Buffer to a File 8-10
Quitting **ed** 8-11

8.4 Displaying Text 8-12

8.5 Locating Lines in a File 8-14
Using Symbols to Identify Lines 8-14
Specifying the Current Line 8-14
Specifying the Last Line 8-15
Specifying the Set of All Lines 8-15
Specifying the Current Line through the Last Line 8-16
Specifying Lines in Relation to the Current Line 8-16

8.6 Displaying Help Messages 8-17

8.7 Creating Text 8-18
Appending Text 8-18
Inserting Text 8-19
Changing Text 8-19
Correcting Typing Errors 8-20

8.8 Undoing the Previous Command 8-21

8.9 Deleting Lines 8-21

8.10 Searching for Text 8-22
Searching for a Pattern of Characters 8-22
Specifying a Global Search 8-24

8.11 Substituting Text 8-25
Substituting on a Single Line 8-25
Substituting on a Range of Lines 8-27
Global Substitution 8-28

8.12 Matching Patterns 8-29
Matching Characters 8-30
Matching the Beginning of a Line 8-31
Matching the End of a Line 8-32
Changing Part of a Line 8-32
Repeating a Substitution Command 8-33
Matching Any Character of a Set 8-33

8.13 Moving Text 8-35
Moving Lines 8-35
Copying Lines 8-36
Joining Lines 8-36
Reading in Another File 8-37

8.14 Miscellaneous Features 8-38
Displaying Nonprinting Characters 8-38
Displaying the Current Filename 8-38
Returning to the Shell 8-39
Recovering From System Interrupts 8-39
Using Markers 8-40

Chapter 8
The ed Line Editor

8.1 Introduction

The **ed** line editor allows you to perform a variety of editing tasks on any type of terminal. You can invoke **ed** commands to display, delete, or change existing lines, to insert new lines, to move or copy lines, and to make substitutions within a line.

This chapter describes **ed** concepts and tells how to use basic **ed** commands. For a technical description of the editor, refer to **ed**(1) in the *Reference Manual.*

8.1.1 Concepts

Before you begin working with **ed**, you should be familiar with the following concepts:

Operating Modes

The **ed** line editor operates in two different modes: input mode and command mode. Input mode allows you to enter text. Command mode allows you to invoke commands to edit or display text. When you create a new file or open an existing file, **ed** is in command mode.

The Buffer

When you use **ed** to create a new file, the text you enter is stored in an editing buffer, which is a temporary work space. Similarly, when you open an existing file, a copy of the file is placed in the buffer, where you can edit it. To save the text in the buffer, you must write the text back to the file. This is accomplished with a simple command.

Line Addresses

Because **ed** operates on lines, it is important to be able to identify lines during an editing session. The editor assigns a number to each line in the buffer. The first line is 1, the second line is 2, and so on. These numbers are called line addresses.

If you want to display or edit a particular line of text, you can move line-by-line through the text until you reach that line, or you can specify the line's address. To do this, enter either the actual number of that line or a character or group of characters that identifies the line. For example, the last line affected by a command is called the current line. To specify this line, you can enter its actual line number or you can type a period, the character that always represents the current line.

When you display lines, you can display the text only, or you can display the text and the associated line numbers.

ed Commands

Most **ed** commands consist of a single character that appears on the screen as you type it. You must press Return to activate these commands.

You can precede an **ed** command with a single line address or a range of line addresses to specify the lines the command is to affect. For example, the **p** command tells the editor to print a line on the screen. If you type **4p**, the editor prints line 4 of the text; if you type **4,10p**, the editor prints lines 4 through 10. If you do not specify a line address, the command affects only the current line.

You can also include **ed** commands in shell programs.

8.1.2 Sample ed Sessions

This section contains two sample **ed** sessions that give you a quick introduction to the **ed** editor. If you would like to start using **ed**, try these examples. Simply type the commands or text described in each step. The commands used in these examples are explained in detail later in this chapter.

Creating a New File

In this example, we will invoke **ed** and create a file named *seminar*, then enter text into the buffer, write the text to a file, and quit **ed**.

1. **Invoke ed.**

   Use this command to start up **ed** and create the *seminar* file:

   $ **ed seminar** Return

   The following display appears:

```
?seminar
```

   The question mark (?) indicates that you are creating a new file.

2. **Invoke input mode.**

   The **ed** editor must be in input mode to allow you to enter text. Use this command to enter input mode:

   a Return

3. **Enter the text into the buffer.**

   Type the following text, pressing Return where indicated.

```
The second seminar in the great books series  Return
will take place Tuesday at 7:00 PM.  Return
Audrey Bryant will be our guest speaker.  Return
```

4. **Return to command mode.**

   When you finish entering text, type a period on a line by itself and press Return. This command returns **ed** to command mode.

```
The second seminar in the great books series
will take place Tuesday at 7:00 PM.
Audrey Bryant will be our guest speaker.
.  Return
```

   Notice that **ed** does not respond to the period; it just waits for another command.

5. **Write the text to a file and quit ed.**

   Enter this command to write the contents of the buffer to a file:

   w Return

   After writing the text to a file, **ed** displays the number of characters the file contains. Now type this command to quit **ed**:

   q Return

   The shell prompt then appears, as shown on the following screen.

```
w  Return
122
q  Return
$
```

Editing an Existing File

In this example, we will use **ed** to edit the *seminar* file created by the first example. We need to change the date of the seminar.

1. **Invoke ed.**

   Use the command shown on the following screen to invoke **ed** and place a copy of the *seminar* file into the buffer. The editor then displays the number of characters the file contains.

```
$ ed seminar  Return
122
```

2. **Display the contents of the file.**

   At this point, **ed** is in command mode and can accept commands to display or edit text. Because *seminar* is a small file, you can display the entire file on the screen at once. To do this, type the following command:

   1,$p Return

   The **1** represents the first line in the file, the **$** represents the last line. The **p** command tells **ed** to print that range of lines.

```
The second seminar in the great books series
will take place Tuesday at 7:00 PM.
Audrey Bryant will be our guest speaker.
```

3. **Move to the line to be changed.**

   Because the final line in the file is the last line affected by the **p** command, that line is the current line. To change the date for the seminar, you must move to the second line. You can use the minus (–) command to move up a line:

   – Return

   The second line of text is now the current line and appears on the screen.

```
The second seminar in the great books series
will take place Tuesday at 7:00 PM.
Audrey Bryant will be our guest speaker.
-  Return
will take place Tuesday at 7:00 PM.
```

4. **Change the date of the seminar.**

   The seminar will now take place on Wednesday. To change the date, type this command:

   s/Tuesday/Wednesday Return

   This command substitutes Wednesday for Tuesday and then prints the new line, as shown on the following screen.

```
s/Tuesday/Wednesday  Return
will take place on Wednesday at 7:00 PM.
```

5. **Write the text back to the file and quit ed.**

   To complete this step, use the **w** and **q** commands as in the first example.

```
w  Return
124
q  Return
$
```

Although these examples show entering text (input mode) and editing text (command mode) in separate editing sessions, you can go back and forth between input and command mode during a single session.

8.2 Invoking ed

To invoke the line editor, type **ed** followed by the name of a new or existing file, then press Return.

If you are creating a new file, the system displays a question mark followed by the filename.

```
$ ed new-file  Return
?new-file
```

If you specify an existing file, **ed** displays the number of characters the file contains.

```
$ ed old-file  Return
235
```

At this point, **ed** is in command mode and can accept commands to edit or display text. If you want to enter new text, you must invoke one of the input mode commands described in "Creating Text," later in this chapter.

8.2.1 Specifying an ed Prompt

When **ed** completes a command, it does not display a prompt to let you know it is ready for a new command. If you would like **ed** to display a prompt, use the **-p** option when you invoke **ed**. You must follow the **-p** option with the character you want **ed** to use as the prompt. For example, the following command tells **ed** to use a colon (:) as a prompt while the *seminar* file is edited.

```
$ ed -p: seminar Return
```

Now **ed** will display a colon (:) when it completes a command, as shown on the following screen.

```
: 2p Return
will take place on Wednesday at 7:00 PM.
:
```

8.3 Writing to a File and Quitting ed

During an editing session, the system holds your text in a temporary work area called a buffer. When you finish editing, you can copy the contents of the buffer into a file. The text in the buffer is not disturbed and can continue to be changed.

If you will be working on the same file for a long period of time, it is a good idea to occasionally write the text in the buffer to a file. If an interrupt occurs (such as an accidental loss of power to your terminal), you might lose the text in the buffer, but you will not lose the copy saved in your file.

8.3.1 Writing the Buffer to a File

The **w** command writes the text in the buffer to a file. You do not need to specify a filename; just type **w** and press Return. The editor writes the text to the file you specified when you invoked **ed**.

If you prefer, you can specify a new name for your file. Be careful not to use the name of an existing file; **ed** will replace its contents with the text in the buffer. **ed** will not warn you that the file exists. It will simply overwrite that file with the contents of the buffer.

To write the text in the buffer to a different file, type **w** followed by the filename. For example, you might edit a file named *memos* and decide to name the edited copy *april.memos*, as shown on the following screen. When the command completes, you can either quit the editor or continue to edit the text in the buffer.

```
w april.memos  Return
85
```

You can also write part of the text in the buffer to a file. To do this, precede the **w** command with a pair of line addresses that specify the lines to write to the file. On the following screen, we entered the command **4,6w meeting** to save lines 4 through 6 in a file called *meeting*.

```
March 17, 1986

Dear Kelly,
There will be a meeting in the
green room at 4:30 P.M. today.
Refreshments will be served.
4,6w  meeting  Return
87
```

8.3.2 Quitting ed

The **q** command allows you to leave the editor and return to the shell. Typically, you will first use the **w** command to save the contents of the buffer. However, if you haven't made any changes to the file, this step is unnecessary. On the following screen, we displayed the contents of the *meeting* file and then quit the editor.

```
$ ed meeting  Return
1,$p  Return
There will be a meeting in the
green room at 4:30 P.M. today.
Refreshments will be served.
q  Return
$
```

If you edited the text in the buffer and want to keep the changes you made, you must write the text to a file before invoking the **q** command. If you type **q** before writing the buffer to a file, **ed** displays a question mark (?), as shown on the following screen. At this point, you can invoke the **w** command to write the text to a file, then type **q** to quit **ed**.

```
q Return
?
w  Return
85
q  Return
$
```

You might not always want to save the text in the buffer. For example, you could make some editing changes to an existing file and then decide you don't want to keep them. If you don't want to save the text in the buffer, invoke the **q** command twice, as shown on the following screen.

```
Refreshments will be served before the meeting.
q Return
?
q Return
$
```

The editor will then discard the contents of the buffer and return you to the shell.

8.4 Displaying Text

ed provides several commands for displaying text on your screen:

p	Displays one or more lines of text
n	Displays one or more lines of text and the line number associated with each line
Return	Displays the next line of text
-	Displays the previous line of text

The **p** command is the default **ed** command. If you type a line address that is not followed by a command, **ed** assumes that you want to display that line on the screen.

To display the current line, type **p** and press Return. To display a different line, precede **p** with the address of that line. For example **2p** prints line 2.

If you want to display a range of lines, type the address of the first line in the range, type a comma, type the address of the last line, and then type the **p** command. For example, the command **2,7p** displays lines 2 through 7, as shown on the following screen. If you specify a range of line addresses, you must type the **p**; otherwise **ed** will display only the last line in the range.

```
2,7p  Return
This is line 2.
This is line 3.
This is line 4.
This is line 5.
This is line 6.
This is line 7.
```

The **n** command displays text and precedes each line with its line number. This command is helpful when you are deleting, creating, or changing lines. The **n** command functions in the same manner as the **p** command. You can type **n** to display the current line or you can precede **n** with a line address or range of line addresses. For example, the command **3n** displays line 3, and the command **2,7n** displays lines 2 through 7, as shown on the following screen.

```
2,7n  Return
2     This is line 2.
3     This is line 3.
4     This is line 4.
5     This is line 5.
6     This is line 6.
7     This is line 7.
```

You can also display the contents of the buffer a line at a time. To display the line below the current line, press Return. That line then appears on the screen and becomes the current line. You can continue to press Return to display each line in turn.

To go backward through the file a line at a time, type a minus sign and press Return. The previous line then appears and becomes the current line.

You can also use the search commands to locate and display a particular line. See "Searching for Text," later in this chapter, for more information.

8.5 Locating Lines in a File

ed uses line numbers to access the text in the buffer. The number of a specific line is always the current position of that line in the buffer. The number assigned to a particular line can change as you revise your text. For example, if you add five lines of text between lines 5 and 6, line 6 becomes line 11. If you delete line 5, line 6 becomes line 5.

When you type an **ed** command, you can precede it with a single line address or a range of line addresses to indicate the lines the command is to affect. If you don't specify a line address, **ed** assumes the command is to affect the current line.

8.5.1 Using Symbols to Identify Lines

The editor uses characters called symbolic addresses to represent specific line addresses. You can use these characters in place of the corresponding line numbers when you invoke **ed** commands.

Specifying the Current Line

The current line is the one most recently acted on by an **ed** command. If you have just invoked **ed** on an existing file, the current line is the last line in the buffer.

A period (.) is the symbolic address for the current line. You can display the current line simply by typing a period and pressing Return, as shown on the following screen. Because we did not specify a command after the period, **ed** executes the default command **p** and displays the current line.

```
This is line 5.
This is line 6.
.  Return
This is line 6.
```

To print the line number assigned to the current line, type this command:

.= Return

ed then prints the line number, as shown on the following screen.

```
This is line 6.
.=  Return
6
```

Specifying the Last Line

The editor recognizes the dollar sign ($) as the symbolic address for the last line in the buffer. When you type a dollar sign, **ed** displays the last line of your text. You can use this character to represent the last line of the buffer in any **ed** command. For example, the command **.,$n** displays all lines and line numbers from the current line (represented by .) through the last line (represented by $).

Specifying the Set of All Lines

The editor recognizes a comma (,) as the symbolic address for all lines in a file, from the first through the last line. It is equivalent to **1,$**. You can use this address to display an entire file, as shown on the following screen.

```
,p  Return
This is line 1.
This is line 2.
This is line 3.
This is line 4.
This is line 5.
This is line 6.
This is line 7.
```

Specifying the Current Line through the Last Line

The semicolon (;) represents all lines beginning with the current line and ending with the last line. It is equivalent to the symbolic address .,$. On the following screen, we used the command **;p** to display all lines in the buffer starting with the current line (the third line in this example).

```
This is line 3.
;p  Return
This is line 3.
This is line 4.
This is line 5.
This is line 6.
This is line 7.
```

8.5.2 Specifying Lines in Relation to the Current Line

You can also specify a line address by adding or subtracting a number of lines from the current line. Addresses derived in this way are called relative addresses.

To specify a line occurring after the current line, type a plus sign (+) followed by the appropriate number of lines. For example, if line 5 is the current line, the command **+4** specifies line 9.

To specify a line occurring prior to the current line, type a minus sign (–) followed by the appropriate number of lines. If line 5 is the current line, the command **–3** displays line 2.

If you ask for a line address that is greater than the last line in the buffer or if you try to subtract a number greater than the current line number, **ed** displays a question mark (?). The current line changes only if you give **ed** a correct address.

You can also use relative line addresses to request a range of lines. The range of relative addresses is calculated from the current line. For example, line 4 is the current line on the following screen. The command **–2,+3n** displays a range of lines beginning two lines before line 4 and ending three lines after line 4.

```
4     This is line 4.
-2,+3n  Return
2     This is line 2.
3     This is line 3.
4     This is line 4.
5     This is line 5.
6     This is line 6.
7     This is line 7.
```

8.6 Displaying Help Messages

If you type an editing command incorrectly or enter a command that **ed** cannot carry out, **ed** prints a question mark (?) on the screen. If you don't know why the question mark appeared, you can issue the help command (**h**), as shown below, to display a short message explaining the problem.

h Return

For example, on the following screen we requested that **ed** print lines 1 through 22. However, because the file contains only four lines, **ed** displayed a question mark (?). When we invoked the **h** command, **ed** printed the message shown on the screen.

```
1  Dear Kelly,
2  There will be a meeting in the
3  green room at 4:30 P.M. today.
1,22p  Return
?
h  Return
line out of range
```

The **H** command causes help messages to print automatically. To issue this command, type **H** and then press Return. The command stays in effect until you either type **H** again or end the editing session. This command can be useful when you are learning the editor.

8.7 Creating Text

The **ed** line editor includes three commands that allow you to create new lines of text:

a	Appends text after the specified line
i	Inserts text before the specified line
c	Changes text

When you issue these commands, **ed** goes into input mode and you can enter as much text as you like. When you finish entering text, type a period on the next line and press Return. The editor will then return to command mode.

When you are working in **ed**, you should be aware that a line of text might not correspond to a physical line on your terminal. The editor considers a line to be all text up to a Return. When the text you are typing reaches the right margin, it wraps around to the next line on the terminal. Therefore, a line in **ed** can include one or more lines on your terminal. To make the lines in **ed** match the physical lines on your terminal, we recommend that you press Return whenever the cursor approaches the right margin.

8.7.1 Appending Text

The **a** command allows you to add text after the current line or after a line address that you specify. To add text after the current line, type this command:

a Return

To add text after a specific address, type the address, then type **a**. For example, the command **4a** adds text after the fourth line in the buffer.

You can also use a symbolic address to specify a particular line. The following command adds text after the last line of the file:

$a Return

When you finish entering text, type a period on the next line and press Return to go back to command mode.

8.7.2 Inserting Text

The **i** command allows you to add text before a specified line in the buffer. It works in the same manner as the **a** command. If you type **i** and press Return, **ed** inserts the text before the current line. If you precede the **i** command with a line address, **ed** inserts the text before that line. For example, if you typed **1i**, the editor would add the text before line 1.

You can also use a symbolic address to specify a line number. For example, the command **$i** inserts text on the line preceding the last line in the buffer.

When you finish entering text, type a period on the next line and press Return to leave input mode.

8.7.3 Changing Text

The **c** command erases the lines you specify and allows you to create one or more lines of text in their place. This command changes the entire line. If you want to change only a few words on a line, use the substitution commands described later in this chapter.

You can use the **c** command to change either the current line or a range of lines. To change the current line, type **c**, press Return, and then type your new text. To change a different line, precede the **c** command with the appropriate line number. For example, your file might look like this:

```
1,$n  Return
1       Line 1
2       Line 2
3       Line 3
4       Line 4
```

To change line 3, you would type the command **3c** and then enter your new text. Because the **c** command puts **ed** into input mode, you can enter as many lines of text as you want. When you finish entering text, type a period on the next line and press Return to leave input mode.

```
5       Line 5
3c  Return
This is the new text
that is replacing line 3.
.   Return
```

The file now looks like this:

```
1,$n  Return
1       Line 1
2       Line 2
3       This is the new text
4       that is replacing line 3.
5       Line 4
```

You can also use the **c** command to change text over a range of lines. To do this, precede the command with a pair of line addresses identifying the first and last lines in the range. For example, the command **8,12c** erases lines 8 through 12 and places **ed** in input mode. You can then enter as much or as little text as you like. The editor will renumber the lines in the buffer as necessary.

8.7.4 Correcting Typing Errors

While **ed** is in input mode, you can use your erase or kill keys to correct typing errors. The erase key deletes the current letter; the kill key deletes the current line.

The default erase key is #, however you can reset it to BackSpace. This key moves the cursor from right to left, deleting each character the cursor crosses. The default key to kill the current line is @. This key can also be reset to another key or key combination such as Ctrl-X. These keys are reset in your *.profile* (or *.login* file for C shell users), as described in Chapter 4.

Occasionally, you might want to include these characters in your text. To keep **ed** from interpreting them as delete commands, precede them with a backslash, for example \@.

8.8 Undoing the Previous Command

The **u** command (short for *undo*) nullifies the last command you invoked to change or delete text. When you invoke a command that deletes or changes text, the original text is stored in a temporary buffer. The **u** command restores the text from the buffer.

To use this command, type **u** and press Return. Because **u** can undo only the previous command, you must enter it immediately after the incorrect command.

The **u** command cannot undo the write (**w**) command or the quit (**q**) command. However, it can undo a previous undo (**u**) command.

8.9 Deleting Lines

The **d** command deletes lines of text. You can delete a single line or a range of lines. To delete the current line, type this command:

d Return

To delete a line other than the current line, precede the **d** command with the appropriate line address. For example, to delete line 14, you would type **14d** and press Return.

If you want to delete a range of lines, precede the **d** command with the line address of the first and last lines in the range. The following command would delete lines 20 through 40:

20,40d Return

You can also use symbolic addresses to identify a line or range of lines for the **d** command. For example, the following command deletes all text from the current line through the end of the buffer.

;d Return

The semicolon (;) is the symbolic address for all lines from the current line through the last line.

8.10 Searching for Text

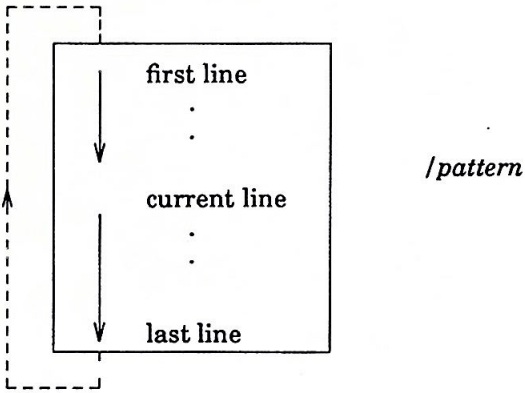
The editor includes several commands that search for a specific pattern of characters. These commands allow you to search for the first occurrence of the pattern, locate all lines containing the pattern, and locate all lines that don't contain the pattern. A search pattern is also called a *regular expression*.

8.10.1 Searching for a Pattern of Characters

The editor can search forward or backward in the buffer to locate a line containing a particular pattern or character string. The pattern can contain more than one word. The search commands are as follows:

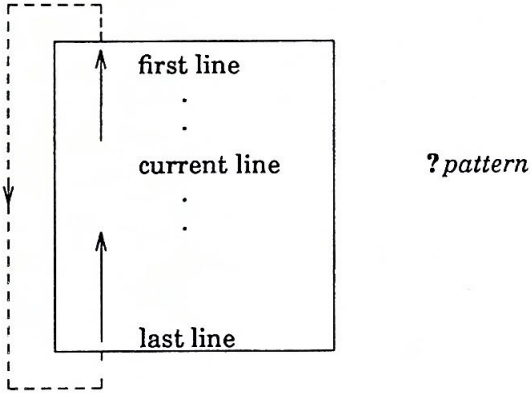
/pattern	Searches forward for the specified *pattern*
?pattern	Searches backward for the specified *pattern*

The following diagram shows the search path **ed** takes when you issue a command in the form */pattern*. The rectangle represents the buffer. **ed** begins at the current line and searches forward (down through subsequent lines in the buffer) for the next line containing the *pattern*. When the search reaches the last line of the buffer, **ed** wraps around to the beginning of the file and continues its search from line 1.



To use this command, type / followed by a character string and press Return. **ed** then searches forward in the file and displays the first line containing the pattern. For example, if you typed the command **/horses** Return, **ed** would display the first line (following the current line) containing the word **horses**. If **ed** cannot find the pattern, it displays a question mark (?).

The next diagram shows the search path **ed** takes when you issue a command of the form *?pattern*. The editor begins at the current line and searches backward (up through previous lines in the buffer) for a line containing *pattern*. If the search reaches the first line of the file, **ed** wraps around to the end of the file and continues searching upward.



To use this command, type ? followed by a character string and press Return. **ed** then searches backward in the file and displays the first line containing that string.

For example, if you typed the command **?lambs** Return, the editor would search backward from the current line and display the first line containing the word `lambs`. If **ed** does not find the pattern, it displays a question mark (?).

You can also use the / and ? commands to continue a search. After **ed** displays the first occurrence of a search pattern, it waits for another command. To continue the search in the same direction, type /, press Return, and the editor will search for the next occurrence of the pattern. To continue the search in the opposite direction, type ? and press Return.

8.10.2 Specifying a Global Search

The **g** and **v** commands perform global searches for a particular pattern of characters. The commands take this form:

g/*pattern*	Searches for all lines containing the pattern
v/*pattern*	Searches for all lines that do not contain the pattern

To show how these commands differ, we will invoke them on this file:

```
1,$p  Return
This is the first line of text.
This is the second line.
This is the third line of text.
This is the fourth line.
```

The command **g/text** prints all lines containing the search pattern **text**:

```
g/text  Return
This is the first line of text.
This is the third line of text.
```

The command **v/text** prints all lines that do not contain the search pattern **text**:

```
v/text  Return
This is the second line.
This is the fourth line.
```

You can follow either of these commands with a slash and another command. For example, you could add **/n** to print line numbers.

```
g/text/n  Return
1       This is the first line of text.
3       This is the third line of text.
```

8.11 Substituting Text

The substitute commands replace a string of characters with new text. These commands are useful when you want to change a few words in a line or a piece of text occurring throughout a file. The substitute commands are as follows:

s Changes the first occurrence of the specified text on the lines you indicate

g Changes all occurrences of the specified text on the lines you indicate

8.11.1 Substituting on a Single Line

To make a substitution on the current line, type a command in one of these forms:

s/*old_text*/*new_text*/ Replaces the first occurrence of *old_text* with the *new_text*

s/*old_text*/*new_text*/**g** Replaces all occurrences of *old_text* with the *new_text*

For example, the line on the following screen contains a typing error. In the substitute command, the old text is the **ai** of `airor`; the replacement text is **er**. When **ed** locates the old text (**ai**), it replaces it with **er**. If **ed** cannot find the specified old text, it displays a question mark (`?`).

```
I made an airor during the game.
s/ai/er  Return
I made an error during the game.
```

Because we omitted the trailing / in this command, **ed** added **/p** by default and printed the corrected line.

However, you can type a different command at the end of the substitution command. For example, if you want **ed** to display the line number when it prints the line, add **n** to the command, as shown on the following screen.

```
This is a test foil.
s/foil/file/n  Return
3        This is a test file.
```

If the line contains more than one occurrence of the old text, you can add **g** to the end of the command:

s/foil/file/g Return

The editor will now replace all occurrences of the pattern. The corrected line will not appear on the screen.

To substitute text on a line other than the current line, precede the substitute command with the address of the line. For example, if line 8 is the current line and you need to substitute text on line 3, you would precede the substitute command with **3**, as shown on the following screen.

```
3s/test/sample/n  Return
3     This is a sample file.
```

You can also use the substitute command to delete text. To do this, type the words to be deleted as the search pattern (*old_text*), and don't enter any replacement text. We used this method to delete the word `sample` from the line shown on the following screen. Notice that we also deleted the space following **sample.**

```
This is a sample file.
s/sample /  Return
This is a file.
```

8.11.2 Substituting on a Range of Lines

You can precede a substitute command with a range of line addresses. The editor will then make the substitution on all lines falling within that range. If a line within the range doesn't contain the specified pattern, **ed** doesn't make any changes to that line.

On the following screen, we specified a range of lines for the substitute command. When **ed** performs a substitution on a range of lines, it prints only the last line changed. In this example, it displays line 4 and its line number because we added **/n** to the command.

```
4     test 1, 2, 3
1,4s/test/TEST/n  Return
4     TEST 1, 2, 3
```

If the search pattern occurs more than once in a line, the **s** command changes only the first occurrence of the pattern. To cause **ed** to change all occurrences of the pattern, add **/g** to the end of the command. The following command will change each occurrence of the pattern:

1,4s/test/TEST/g Return

When you add **/g** to a substitution command, **ed** does not display lines when it performs the substitution. To verify the substitution, use a **p** or **n** command.

8.11.3 Global Substitution

You can use global substitution to make a change throughout an entire file. The editor replaces each occurrence of the search pattern with the new text. The global substitution command takes this form:

g/*old_text*/**s//***new_text*/**g**

Be sure to type two slashes (//) after the **s**, and keep in mind that **ed** distinguishes between upper- and lowercase letters. You must press Return to activate this command.

The following command changes all occurrences of the word `exercise` to `practice`:

g/exercise/s//practice/g Return

8.12 Matching Patterns

When you are using search or substitute commands, you can include the following special characters in search patterns. (Search patterns are also called *regular expressions*.) The characters act as wild cards for matching characters.

.	Matches one character
*	Matches zero or more occurrences of the preceding character
.*	Matches zero or more occurrences of any character following the period
^	Matches the beginning of the line
$	Matches the end of the line
&	Repeats the last replacement pattern
[...]	Matches the first occurrence of a character in the brackets
[^...]	Matches the first occurrence of a character that is not in the brackets

When you use a special character in a search pattern, **ed** matches the longest possible string starting as close to the beginning of the line as possible.

If you want to use a special character as text in a search pattern, you must tell **ed** to interpret the character literally. To do this, precede the character with a backslash. For example, to use an asterisk (*) as text, you would type *****. If you want **ed** to interpret a backslash (\) literally, you must precede it with a backslash (\\).

Matching Characters

A period (.) in a search pattern matches any character. For example, your file might contain these lines:

```
1,$p  Return
rat
cat
turtle
at
cow
goats
```

The global search command **g/.at** causes **ed** to search for any three-character sequence ending in the pattern **at**. The sequence can be part of a longer word.

```
g/.at  Return
rat
cat
goats
```

The * matches zero or more occurrences of the preceding character. For example, the search pattern **ca*t** matches c, followed by zero or more occurrences of a, followed by t. This search pattern matches strings such as `cat`, `caat`, and `ct`.

If you combine the period (.) and the asterisk (*), **ed** will match all characters. You can use this combination to replace the characters in the last part of a line. On the following screen, the substitution pattern **are.*** represents all characters from `are` through the end of the line. These characters are replaced by the new text.

```
Toads are very cold creatures
s/are.*/are wonderful and warm  Return
Toads are wonderful and warm
```

The **.*** combination can also replace all characters between two patterns. On the following screen, the search pattern **are.*cold** represents all characters from `are` through `cold`. The new text replaces these characters.

```
Toads are very cold creatures
s/are.*cold/are wonderful and warm  Return
Toads are wonderful and warm creatures
```

Matching the Beginning of a Line

The caret (^) matches the beginning of a line. If you want to insert a word at the beginning of a line, type a ^ in the *old_text* position of the substitute command. Then type the words to be inserted in the *new_text* position. Be sure to leave a space after the new text you're adding. The next example places the word **all** (followed by a space) at the beginning of each line.

```
1,$s/^/all /  Return
1,$p  Return
all creatures great and small
all things wise and wonderful
all things bright and beautiful
```

Matching the End of a Line

The dollar sign ($) character matches the end of a line. It can be used to add characters at the end of a line or range of lines. Place the dollar sign ($) in the *old_text* position of the substitute command, then type the text you are adding in the *new_text* position. Be sure to leave a space before the new text, as shown on the following screen.

```
Daniel rode his bike
s/$/ to the top of the hill  Return
Daniel rode his bike to the top of the hill
```

You can also use the $ character to add punctuation to the end of a line. Because the period in the following example is in the *new_text* position rather than in the search pattern, it does not have a special meaning.

```
Daniel rode his bike to the top of the hill
,s/$/.  Return
Daniel rode his bike to the top of the hill.
```

Changing Part of a Line

If you want to insert text into the middle of a line, type an ampersand (&) as the first character of the replacement pattern, then type a space followed by the replacement text. The ampersand (&) tells **ed** to insert the replacement text between the search pattern and the rest of the line. For example, on the following screen **ed** inserts the new text after the search pattern (`thal`) and retains the rest of the original line.

```
The neanderthal skeletal remains
s/thal/& man's/p  Return
The neanderthal man's skeletal remains
```

Repeating a Substitution Command

The percent sign (%) repeats a substitution command. This feature allows you to perform the same substitution on several lines without retyping the command. When you want to repeat the previous substitution command on another line, type the line address, then type **s//%**. For example, to repeat your last substitution command on line 15, you would type this command:

15s//% Return

Matching Any Character of a Set

You can use brackets to define a set of characters for a search pattern. The editor then matches the first occurrence of any of the characters enclosed in brackets. In the following example, the search pattern is **[6789]**. When **ed** locates one of these numbers, it changes the number to 4, the specified replacement text.

```
1,$p  Return
Monday       33,000
Tuesday      75,000
Wednesday    88,000
Thursday     62,000
1,$s/[6789]/4/g  Return
1,$p  Return
Monday       33,000
Tuesday      45,000
Wednesday    44,000
Thursday     42,000
```

You can also use brackets to delete any of a set of characters. The following example deletes Mr or Ms from a list of names. The pattern **M[rs]** tells **ed** to locate the first occurrence of M followed by either r or s. When **ed** finds this pattern, it deletes the characters and the following space because we did not specify any replacement text.

```
Mr Arthur Middleton
Mr Matt Lewis
Ms Anna Kelley
1,$s/M[rs] //  Return
,$p  Return
Arthur Middleton
Matt Lewis
Anna Kelley
```

If a caret (^) is the first character in brackets, **ed** interprets it as an instruction to match all characters not within the brackets. However, if the caret is in any other position within the brackets, **ed** interprets it literally, as a caret.

On the following screen, the search pattern **grade [^AB]** tells **ed** to locate the first occurrence of "grade" that is not followed by A or B. The editor then replaces this occurrence with **grade A**. Because we preceded the substitute command with line addresses specifying the entire file, **ed** makes the substitution on each line containing the search pattern.

```
grade A   Computer Science
grade C   Robot Design
grade B   Jogging
grade C   Tennis
1,$s/grade [^AB]/grade A  Return
1,$p  Return
grade A   Computer Science
grade A   Robot Design
grade B   Jogging
grade A   Tennis
```

When you include special characters in a pattern, be sure the pattern will match the characters you want. In the above example, if you had specified the substitution pattern **1,$s/[^AB]/A**, the editor would match g, the first character in each line, and change it to A.

8.13 Moving Text

The editor includes commands that allow you to move, copy, or join lines of text in the editing buffer. You can also read text from another file into the editing buffer.

8.13.1 Moving Lines

The **m** command moves blocks of text to another location in the file. You can move a single line or a range of lines.

To move the current line to another location, type **m** followed by the address of the line preceding the new location. **ed** will then insert the current line after that line. For example, the following command moves the current line to the position after line 4:

m4 Return

To move a different line, precede the **m** command with the address of that line. The following command moves the last line in the file to the position after the first line:

$m1 Return

If you want to move a range of lines, precede the **m** command with the addresses for the first and last lines in the range. For example, the following command moves lines 8 through 12 below line 65. When **ed** moves the text, it will renumber the lines.

8,12m65 Return

To move one or more lines to the beginning of the file, follow the **m** command with zero. For example, the following command moves lines 3 and 4 to the beginning of the file.

3,4m0 Return

8.13.2 Copying Lines

The **t** command copies the lines you specify and places them in another location. It does not delete the original lines.

The **t** command works in the same manner as the **m** command. To copy the current line, type **t** followed by the address of the line preceding the location for the copied line. For example, the following command makes a copy of the current line and places the copy after line 4.

t4 Return

To copy a different line, precede the **t** command with the line address of that line. The following command makes a copy of line 8 and places it after the last line in the file.

8t$ Return

You can also copy a range of lines by preceding the **t** command with the addresses of the first and last lines in the range. The following command makes a copy of lines 4 through 8 and places the copied lines after line 22.

4,8t22 Return

To copy one or more lines to the beginning of the file, follow the **t** command with zero. For example, the following command makes a copy of lines 5 and 6 and places the copy at the beginning of the file.

5,6t0 Return

8.13.3 Joining Lines

The **j** command joins the current line with the line that follows it. To use this command, type **j**, press Return, and the editor will join the two lines.

For example, you might want to join these two lines:

```
10    Now is the time to join
11    the team.
```

To do this, first use the **p** or **n** command to make line 10 the current line, then invoke the **j** command.

```
10n  Return
10      Now is the time to join
j Return
```

The line now looks like this:

```
10n  Return
10      Now is the time to jointhe team.
```

Notice that **ed** did not insert a space where the two lines joined. You can use the **s** command to insert the space.

s/jointhe/join the Return

8.13.4 Reading in Another File

The **r** command copies text from another file into the buffer. You can add the text at the end of the buffer or after a line that you specify. To add the text at the end of the buffer, type **r** followed by the name of the file to be copied into the buffer. The following command places a copy of the *memos* file at the end of the buffer.

r memos Return

If you want to append a copy of the file after a specific line in the buffer, type the line address of that line, then type **r** followed by the filename. For example, to place a copy of the *memos* file after line 12, type this command:

12r memos Return

After adding the file to the buffer, **ed** prints the number of characters the file contains.

8.14 Miscellaneous Features

This section describes several commands and features you might find useful during editing sessions.

8.14.1 Displaying Nonprinting Characters

When you are using **ed**, many characters such as tabs and control characters do not print on your terminal. The **l** command causes surrogate characters to print in place of non-printing characters. For example, tabs are indicated by a greater-than sign (>) and control characters by their octal values.

To invoke this command on the current line, type **l** and press Return. You can also precede the command with a line address or range of line addresses to specify the lines you want the **l** command to affect.

8.14.2 Displaying the Current Filename

The **f** command prints the name of the file currently in the buffer. You can also use this command to give the contents of the buffer a new name.

To display the name of the file in the buffer, type this command:

f Return

To assign a new filename to the text in the buffer, type **f** followed by the new filename and then press Return. For example, if you are editing the *memos* file and decide to rename the text in the buffer *paula.memos*, you would type the command shown on the following screen.

```
f paula.memos  Return
paula.memos
```

When you use the **w** command to write the text in the buffer to a file, the text will be written to the *paula.memos* file. You can assign a new filename to the text in the buffer at any time during the editing session.

8.14.3 Returning to the Shell

The exclamation mark (!) allows you to return to the shell, execute a shell command, and then go back to the current line in the buffer.

To use this command, type an exclamation mark (!) as the first character on a line, then type the shell command. The command output will appear on the screen or go to a file as you directed. When the command finishes executing, **ed** prints an exclamation mark on the next line. You are now back at the current line in the buffer.

For example, if you want to return to the shell to find out the correct date, type **!** and the shell command **date**, as shown on the following screen. The system then displays the date, followed by a ! on the next line.

```
This is the current line
! date [Return]
Tue  Apr 1  14:24:22  EST  1986
!
```

8.14.4 Recovering From System Interrupts

If a system interrupt occurs while you are using **ed**, the operating system tries to save the contents of the editing buffer in a file named *ed.hup*. Later you can use one of the following methods to retrieve your text from this file.

- Use the **mv** command to change *ed.hup* to another filename. For example, you could delete the original file, then give *ed.hup* that filename.
- Invoke **ed** on the *ed.hup* file, then use the **f** command to rename the contents of the buffer. If you use this method to recover the contents of the buffer, be sure to remove the *ed.hup* file afterward.

8.14.5 Using Markers

When you are using **ed**, you can set markers anywhere in your text. You can use up to 26 markers; the markers are named a through z. The markers you set stay in effect until you either reset them or finish the editing session.

To set a marker, type **k** followed by the name of the marker. For example, to set marker **r**, you would type **kr** and press Return.

When you want to return to a marker, type ' followed by the name of the marker. To display the line containing marker r, you would type this command:

'r Return

You can also use markers to specify line addresses. For example, to change the word "honey" to "sugar" starting at marker m and going to marker n, you would type this command:

'm,'ns/honey/sugar/g Return

Chapter 9
The vi Screen Editor

9.1 Introduction 9-1
Concepts 9-1
vi Commands 9-1
Line Editor Commands 9-1
vi Operating Modes 9-2
The Buffer 9-2
The Shell Environment 9-2
Sample **vi** Sessions 9-3
Creating a New File 9-3
Editing an Existing File 9-5

9.2 Invoking vi 9-9
Recovering a File Lost by an Interrupt 9-10
Editing Multiple Files 9-10
Viewing a File 9-11

9.3 Writing to a File and Quitting vi 9-11
Writing and Quitting 9-12
Writing to a File 9-13
Quitting **vi** 9-14

9.4 Creating Text 9-14
Invoking Input Mode 9-14
Entering Text 9-15
Correcting Errors 9-16

9.5 Moving the Cursor 9-17
Basic Cursor Movement 9-17
Positioning the Cursor on a Character 9-18
Moving the Cursor Right or Left 9-18
Moving the Cursor to the Beginning or End of a Line 9-19
Searching for a Character on a Line 9-19
Positioning the Cursor by Lines 9-20
Moving Up a Line 9-20
Moving Down a Line 9-20
Positioning the Cursor by Words 9-21
Positioning the Cursor by Sentences 9-23

Positioning the Cursor by Paragraphs 9-23
Positioning the Cursor on the Screen 9-23
Scrolling Text 9-24
Scroll Forward a Full Screen 9-25
Scroll Forward a Half Screen 9-25
Scroll Backward a Full Screen 9-25
Scroll Backward a Half Screen 9-25
Line Numbers 9-26
Displaying Line Numbers 9-26
Moving to a Specific Line 9-27
Searching for a Pattern of Characters 9-28
Using Special Characters in a Search Pattern 9-29

9.6 Undoing Commands 9-30

9.7 Deleting Text 9-30
Deleting Characters 9-30
Deleting Words 9-31
Deleting Lines 9-32
Deleting the Rest of the Line 9-33
Deleting Sentences 9-33
Deleting Paragraphs 9-33
Deleting the Rest of the Buffer 9-34

9.8 Modifying Text 9-34
Replacing Text 9-34
Substituting Text 9-36
Changing Text 9-37
Changing Words 9-37
Changing Lines 9-38
Changing Sentences 9-39
Changing Paragraphs 9-39
Making Global Changes 9-40

9.9 Cutting and Pasting Text 9-41
Pasting Text 9-41
Fixing Transposed Letters 9-42
Copying Text 9-42
Using Registers to Copy or Move Text 9-43

9.10 Miscellaneous Commands 9-44
Repeating the Last Command 9-44
Joining Two Lines 9-45
Clearing and Redrawing the Window 9-45
Changing Letter Case 9-45

Adding a File to the Buffer 9-45
Returning to the Shell 9-46
Using Markers 9-46
Locating Spelling Errors 9-47

9.11 Modifying Your vi Environment 9-48
Displaying Parameters 9-48
Changing Parameters 9-48
Setting an Automatic Return 9-49
Displaying Line Numbers 9-49
Displaying Input Mode 9-50
Ignoring Case in Searches 9-50
Displaying Tabs and Marking Ends of Lines 9-50
Setting Automatic Indentation 9-50
Setting Shiftwidth 9-50

Chapter 9
The vi Screen Editor

9.1 Introduction

The **vi** editor is a powerful tool for creating and editing files. It is designed for use with a video display terminal, allowing you to create and edit text on the screen.

This chapter describes **vi** concepts and tells how to use basic **vi** commands. For a technical description of the editor, refer to **vi**(1) in the *Reference Manual.*

9.1.1 Concepts

Before you begin working with **vi**, you should be familiar with the following concepts.

vi Commands

Most **vi** commands consist of a single uppercase or lowercase letter, or a combination of Ctrl and a letter. Most of these commands are silent; they do not appear on the screen as you type them. You do not need to press Return after entering a **vi** command. The command takes effect immediately.

Case is important when entering **vi** commands. Frequently, the uppercase and lowercase version of the same letter represent different commands. If commands are not operating as you expect, make sure that CapsLock is in the unlocked position.

Line Editor Commands

The **vi** editor can also accept many **ex** and **ed** line editor commands. The **ex** commands are similar to the **ed** commands discussed in the previous chapter. (For a list of **ex** commands, refer to **ex**(1) in the *Reference Manual.*)

Line editor commands begin with a colon that appears at the bottom of the screen when you type it. The remainder of the command also appears at the bottom of the screen. You must press Return to activate these commands.

vi Operating Modes

The **vi** editor operates in two different modes: input mode and command mode. Input mode allows you to enter text. Command mode allows you to edit text or move the cursor. When you create a new file or open an existing file, the editor is in command mode. You must invoke an input command to enter input mode. To return to command mode, press Esc.

The characters you type should appear on the screen only when **vi** is in input mode or when you are entering line editor commands (which appear at the bottom of the screen). If you are entering **vi** commands and letters appear on the screen, you are still in input mode. Press Esc to return to command mode and try the commands again.

The Buffer

When you use **vi** to create a new file, the text you enter is stored in an editing buffer, which is a temporary work space. Similarly, when you open an existing file, a copy of the file is placed in the buffer where you can edit it. To save the text in the buffer, you must write it to a file. This is accomplished with a simple command.

The Shell Environment

When you log into the operating system, the shell sets up an environment for you to work in. This environment specifies values for variables such as **PATH**, which defines your search path for commands, and **HOME**, which stores the name of your home directory. Chapter 4 describes these variables and tells how to change them.

For **vi** to operate properly on your terminal, the **TERM** environment variable must be set properly. This variable specifies the type of terminal you are using. Because **vi** executes differently on different terminals, it needs this information to function correctly. Typically, the system administrator sets this value for the TTY line you are using. If a message such as `unknown terminal type` appears when you try to use **vi**, contact your system administrator for assistance.

You can also create a file called *.exrc* to store parameters that affect how **vi** operates. For example, you can set a parameter that causes **vi** to display line numbers on the screen. The section "Modifying Your **vi** Environment," at the end of this chapter, describes these parameters.

9.1.2 Sample vi Sessions

This section contains two sample **vi** sessions that will give you a quick introduction to the **vi** editor. If you would like to start using **vi**, try these examples. Simply type the commands or text described in each step. The commands used in these examples are explained in detail later in this chapter.

Creating a New File

In this example, we will invoke **vi** and create a file named *meeting*, then enter text into the buffer, write the text to a file, and quit **vi**.

1. **Invoke vi.**

   Use this command to start up **vi** and create the *meeting* file:

   $ **vi meeting** Return

   The following display then appears on the screen.

```
_
~
~
~
~
"meeting" [New file]
```

   The underscore shows the cursor position; the tildes (~) mark every other blank line. The message at the bottom of the screen tells the name of the file and indicates that the file has just been created.

2. **Invoke input mode.**

   The **vi** editor must be in input mode to allow you to enter text. Invoke this command to enter input mode:

   a

   The command does not appear on the screen.

3. **Enter the text into the buffer.**

   Type the following text, pressing Return when indicated to go to the next line on the screen.

```
There will be a meeting to discuss  Return
moving our offices to our new location.  Return
The meeting is scheduled for March 29th  Return
at 3:00 PM.

~
~
~
```

4. **Return to command mode.**

   When you finish entering text, press Esc to cause **vi** to leave input mode and return to command mode.

```
There will be a meeting to discuss
moving our offices to our new location.
The meeting is scheduled for March 29th
at 3:00 PM. Esc

~
~
```

5. **Write the text to a file and quit vi.**

   Enter this command to write the contents of the buffer to a file and then leave **vi**:

   ZZ

   The **ZZ** command does not appear on the screen. A message appears at the bottom of the screen telling you the name of the file and the number of lines and characters it contains. The shell prompt ($) follows the message.

```
There will be a meeting to discuss
moving our offices to our new location.
The meeting is scheduled for March 29th
at 3:00 PM.

~
~
~
"meeting" [New file] 4 lines, 127 characters
$
```

Editing an Existing File

In this example, we will use **vi** to edit the *meeting* file created by the first example. We are going to change the date of the meeting from March 29th to April 3rd. To make this change, complete the following steps:

1. **Invoke vi.**

   Use the following command to invoke **vi** and place a copy of the *meeting* file into the buffer:

   $ **vi meeting** Return

   When the text appears on the screen, **vi** will be in command mode, allowing you to move the cursor and enter editing commands. Right now, the cursor is on the first character in the first line, as indicated by the underscore.

```
There will be a meeting to discuss
moving our offices to our new location.
The meeting is scheduled for March 29th
at 3:00 PM.

~
~
```

2. **Move the cursor.**

   Before you can change the date for the meeting, you must move the cursor to `March`. To do this, first type the **j** command twice to move the cursor down two lines. Then type the command **5w** to move the cursor 5 words to the right.

   The cursor is now on the first character of `March`, as shown on the screen in Step 3.

3. **Edit the text.**

   You can now change the date from `March 29th` to `April 3rd`. To do this, type the following command, which allows you to change two words:

   2cw

   A dollar sign (`$`) appears in place of the last character of `29th`. This sign indicates the last character of the existing text that will be affected by this editing command.

```
There will be a meeting to discuss
moving our offices to our new location.
The meeting is scheduled for March 29t$
at 3:00 PM.

~
~
```

The **2cw** command places **vi** in input mode, allowing you to enter as much text as you like. Type the new date, `April 3rd`, and press Esc to return to command mode. The screen now looks like this:

```
There will be a meeting to discuss
moving our offices to our new location.
The meeting is scheduled for April 3rd
at 3:00 PM.

~
~
```

4. **Write the text to a file and quit vi.**

   Now write the contents of the buffer back to the *meeting* file and quit **vi**. In the first example, you used the **ZZ** command to do this. This time, try the following line editor command to perform these tasks:

   :wq Return

   Because **:wq** is a line editor command, it begins with a colon and appears on the bottom of the screen as you type it. You must press Return to activate the command.

```
There will be a meeting to discuss
moving our offices to our new location.
The meeting is scheduled for April 3rd
at 3:00 PM.

~
~
:wq Return
```

The **ZZ** and **:wq** commands produce identical results. They both write the contents of the buffer to a file and quit **vi**. A message then apepars stating the name of the file and the number of lines and characters it contains. The shell prompt (`$`) appears below the message.

Although these examples demonstrate how to enter text (input mode) and edit text (command mode) in separate **vi** sessions, you can go back and forth between input mode and command mode during a single **vi** session.

9.2 Invoking vi

To begin using the editor, type **vi** and the name of the file you want to create or edit. For example, to create a file named *practice*, you would type this command:

```
$ vi practice Return
```

vi then clears the screen and displays a window in which you can enter and edit text.

```
_
~
~
~
~
~
~
~
"practice" [New file]
```

The underscore (_) on the top line shows the cursor waiting for you to enter a command. (On your terminal, the cursor might be a blinking underscore or a reverse color block.) Every other line is marked with a tilde (~), the symbol for an empty line.

If you want to edit an existing file, type **vi** followed by the name of the file. For example, to edit the *memos* file, you would type this command:

```
$ vi memos Return
```

The system then places a copy of the *memos* file into the **vi** buffer and the file appears on your screen.

After creating a new file or opening an existing file, **vi** is in command mode. If you want to enter text, you must invoke input mode, as described in "Creating Text," later in this chapter.

When invoking the **vi** editor, you can instruct **vi** to perform these tasks:

- Recover a file lost by an interrupt to the operating system.
- Place several files in the editing buffer and edit each in turn.
- View a file in read-only mode.

9.2.1 Recovering a File Lost by an Interrupt

If the system experiences an interrupt or disconnect, it will exit **vi** without writing the text in the buffer to a file. However, the operating system will store a copy of the buffer for you. When you log back into the system, you can use the **–r** option to the **vi** command to restore the file. For example, to restore the *memos* file, you would type this command:

vi –r memos Return

The changes you made to *memos* before the interrupt occurred are now in the **vi** buffer. You can continue editing the text, or you can write the buffer to a file and quit **vi**.

9.2.2 Editing Multiple Files

If you want to edit more than one file in the same editing session, specify each filename when you issue the **vi** command. For example, if you want to edit both the *memos* and the *trips* files, you would type this command:

vi memos trips Return

vi then displays a message similar to the following one that specifies how many files you are going to edit.

```
2 files to edit
```

After you have edited the first file, use the **:w** command to write the contents of the buffer to a file.

:w Return

The system then displays the name of the file and the number of lines and characters it contains. Now type the **:n** command to bring the next file into the editing buffer and display its contents on the screen.

:n Return

9.2.3 Viewing a File

Because the **vi** editor provides powerful search and scroll capabilities, it is an excellent tool for inspecting files. However, you might want to protect yourself from accidentally changing a file during a **vi** session. The **vi** read-only option prevents you from writing in a file. To use this option, invoke the editor as **view** rather than **vi**. For example, to scan the *memos* file in read-only mode, you would enter this command:

```
$ view memos Return
```

9.3 Writing to a File and Quitting vi

When you finish editing a file, **vi** gives you many options for saving your text and exiting the editor. The following command sequences allow you to save or discard the contents of the buffer, and to continue to work in **vi** or return to the shell. (These commands are explained in detail later in this section.) Commands preceded by a colon are line editor commands and must be followed with Return.

ZZ or **:wq**	Writes the contents of the buffer to the file specified when **vi** was invoked and then quits **vi**
:w **:e***filename*	Writes the contents of the buffer to a file, then copies another file into the buffer
:w *filename* **:q**	Writes the contents of the buffer to a new file named *filename* and then quits **vi**
:w! *filename* **:q**	Overwrites an existing file called *filename* with the contents of the buffer and then quits **vi**
:q!	Discards the contents of the buffer and quits **vi** (use when the buffer has been edited)
:q	Quits **vi** without writing the buffer to a file (use when the buffer has not been edited)

9.3.1 Writing and Quitting

Both the **ZZ** and **:wq** commands write the contents of the buffer to a file, quit **vi**, and then return you to the shell.

The **ZZ** command causes the editor to move the text in the buffer to the file you specified when you invoked **vi**. It also displays the name of the file and the number of lines and characters it contains, followed by the shell prompt.

```
I want to write this file
and return to the shell.  ZZ
~
~
"practice" [New file] 7 lines, 151 characters
$
```

You can also use the **:w** and **:q** line editor commands to write and quit a file. The **:w** command writes the buffer to a file; the **:q** command quits **vi** and returns you to the shell. You can type these commands separately or combine them into the single command **:wq**, as shown on the following screen. This command sequence writes the buffer to the file you specified when you invoked **vi**.

```
I want to write this file
and return to the shell.
~
~
:wq  Return
```

The name of the file and the number of lines and characters it contains then appear on the screen, followed by the shell prompt ($).

9.3.2 Writing to a File

The **:w** command writes the contents of the buffer to a file. If you are working on a file for a long period of time, it's a good idea to occasionally write the buffer to a file. The text in the buffer is unaffected by this step and can still be edited.

To write the text in the buffer to the file you specified when invoking **vi**, simply type **:w** and press Return.

You can also write the contents of the buffer to a new file. For example, you might want to write the buffer to a new file called *memos*. To do this, you would type this command:

:w memos Return

If the command creates the file successfully, a message such as the following will appear:

```
"memos" [New file] 80 lines, 20 characters
```

If the file you specified already exists, you will receive a warning. For example, if the *memos* file already exists, this message will appear:

```
"memos" File exists - use "w! memos" to overwrite
```

If you want to replace the contents of the existing *memos* file with the contents of the buffer, use the **:w!** command to overwrite *memos*.

:w! memos Return

The **:w** command can also write a portion of the text in the buffer to a file. To do this, specify the appropriate line number or range of line numbers followed by the name of the new file. (See "Line Numbers," later in this chapter, for information about displaying line numbers.) For example, to write the third line of the buffer to a file named **three**, type this command:

:3w three Return

To specify a range of lines, type the line number assigned to the first line in the range, type a comma, then type the number assigned to the last line. The following command writes lines 23 through 37 to a file named *letter*:

:23,37w letter Return

After writing the buffer to a file, you can use the **:q** command described in the next section to quit **vi**, or you can invoke the **:e** command to edit another file. The **:e** command tells **vi** to copy the file you specify into the buffer. The following command sequence writes the buffer to a file, then brings the *promotions* file into the buffer for editing:

:w Return
:e promotions Return

9.3.3 Quitting vi

The **:q** command quits the **vi** editor. To use this command, you must first write the buffer contents to a file or have made no changes to the file. If you edited the text before invoking **:q**, **vi** will display a message telling you either to save the contents of the buffer (by invoking **:w**) or to discard the buffer, as described below.

You might not always want to write the buffer to a file. For example, if you make some changes to a file and then decide you don't want to keep the changes, you can discard the text in the buffer and then leave **vi**. To do this, type this command:

:q! Return

9.4 Creating Text

When **vi** is in input mode, you can enter text into the buffer. This section describes the commands that place **vi** in input mode and tells how to enter text and correct errors.

9.4.1 Invoking Input Mode

The **vi** editor provides several commands that you can use to place **vi** in input mode. However, the commands differ in where **vi** inserts the text.

a	Appends text after the cursor
A	Appends text at the end of the current line
i	Inserts text before the cursor
I	Inserts text at the beginning of the current line
o	Opens a new line below the current line
O	Opens a new line above the current line

The **i** command inserts text to the left of the cursor position. The **a** command adds text to the right of the cursor. For example, your file might contain the following line:

```
a wind came up early that day
```

If the cursor is on the first character of the line and you use **a** to invoke input mode, **vi** appends the characters you type after the first character:

```
a strong wind came up early that day
```

If you use **i** to invoke input mode, **vi** inserts the characters you type before the first character:

```
I remember a wind came up early that day
```

The **A** and **I** commands operate similarly. **A** adds text at the end of the current line; **I** inserts text at the beginning of the current line. If the line begins with a space, **vi** inserts the text after the space.

The **O** command creates a line directly above the current line; the **o** command creates a line below the current line. The cursor then waits at the beginning of the new line for you to enter text. You can issue these commands from any point in the current line.

9.4.2 Entering Text

The **vi** editor considers a line to be a string of characters bounded on either side by a newline indicator such as Return. A line in **vi**, therefore, doesn't necessarily correspond to a physical line on your terminal. When the text you are typing reaches the right side of the screen, it wraps around to the next physical line on the screen. Yet, **vi** still considers this a single line. We recommend that you press Return whenever the cursor approaches the right side of the screen. This will make editing easier, as each line in **vi** will correspond to a physical line on the screen.

You can have **vi** insert a Return for you automatically whenever the cursor approaches the right side of the screen. The section "Modifying Your **vi** Environment," at the end of this chapter, tells how to set up this feature, which is called *automatic return*.

If you want to create a practice file to try out the commands described in this chapter, enter about 40 lines of text, leaving a blank line between paragraphs. If you notice a mistake in the line you are typing, use the error correction methods described in the next section. Don't worry about any other mistakes in your text. The remaining sections of this chapter will show you how to correct them.

After you finish typing text, press Esc to leave input mode and return to command mode. If you press Esc and a bell sounds, you are already in command mode. When **vi** is in command mode, you can use the editing commands described later in this chapter to edit the text, or you can write the text in the buffer to a file.

Frequently you will need to insert text into an existing block of text. To do this, move the cursor to the appropriate location by using the cursor movement commands described later in this chapter. Then type an input command and start entering text. As you type, the new text will appear on the screen. All existing characters to the right of the cursor will move to the right to make room for your new text. The **vi** editor will continue to accept characters until you press Esc. If necessary, the original characters will wrap around onto the next line.

9.4.3 Correcting Errors

While **vi** is in input mode you can use your erase key, kill key, or Ctrl-W to correct typing errors. The erase key deletes the current letter, Ctrl-W deletes the current word, and the kill key deletes the current line.

The default erase key is #. However, this key is frequently reset so that you can use BackSpace to erase characters. The default key to kill the current line is @. This key can also be reset to another key or key combination such as Ctrl-X. These keys are reset in your *.profile* (or *.login*, for C shell users), as described in Chapter 4. The following paragraphs assume that BackSpace is the erase key and @ is the kill key.

BackSpace deletes each character the cursor backs across. However, the characters are not erased from the screen until you either type over them or press Esc to return to command mode.

When you type Ctrl-W, the cursor backs across the last word you typed and waits on the first character. It does not erase the word until you either press Esc or type new characters over the old ones.

@ behaves in a similar manner. It removes text entered on the current line during this input mode session. If you are using @ as the kill key and want to type @ as text, precede it with a backslash (\@).

9.5 Moving the Cursor

The **vi** editor includes several commands that move the cursor. These commands allow you to position the cursor in the following ways:

- Move by characters, words, lines, sentences, or paragraphs.
- Scroll text.
- Move to a specific line number.
- Search for a pattern of characters.

To use these commands, **vi** must be in command mode.

9.5.1 Basic Cursor Movement

The following commands move the cursor left, right, up, and down:

h	Moves the cursor one character to the left
j	Moves the cursor down one line
k	Moves the cursor up one line
l	Moves the cursor one character to the right

The **j** and **k** commands maintain the column position of the cursor. For example, if the cursor is on the seventh character from the left, the **j** or **k** command will move the cursor to the seventh character on the new line. If there is no seventh character on the new line, the cursor moves to the last character.

You might find it helpful to mark these four keys with arrows showing the direction in which each key moves the cursor.

NOTE

Some terminals have special cursor control keys marked with arrows. These keys have the same functions as the **h**, **j**, **k**, *and* **l** *commands.*

To move several lines or characters, you can enter the **h**, **j**, **k**, or **l** command several times, or you can precede the command with the number of charac-

ters or lines to move. For example, to move two characters to the right, you can either type **l** twice or type **2l**. To move up four lines, you can either type **k** four times or type **4k**. If the cursor cannot go farther in the direction you requested, **vi** sounds a bell.

9.5.2 Positioning the Cursor on a Character

In **vi**, a character is any letter, number, or symbol appearing in the text. It can be visible or not, printable or not. Spaces and tabs are considered to be characters.

There are three ways to position the cursor on a specific character:

- Move the cursor right or left to the character.
- Move the cursor to the character at either end of the line.
- Search for a character on the line.

The following sections describe each of these methods.

Moving the Cursor Right or Left

As described earlier, the **h** and **l** commands move the cursor right or left to a character on the current line. You can also use Spacebar and BackSpace to move the cursor right or left.

Spacebar	Moves the cursor one character to the right
BackSpace	Moves the cursor one character to the left

You can precede these commands with the number of characters you want the cursor to move. For example, to move the cursor five characters to the right, you would type this command:

5 Spacebar

Similarly, to move the cursor five characters to the left, you would type this command:

5 BackSpace

Moving the Cursor to the Beginning or End of a Line

The following commands move the cursor to the first or last character of a line:

$	Moves the cursor to the last character of a line
0 (zero)	Moves the cursor to the first character of a line
^	Moves the cursor to the first nonblank character of a line

Searching for a Character on a Line

The **f**, **F**, **t**, and **T** commands search for a specific character on a line. You must follow these commands with the character **vi** is to search for. If **vi** doesn't find the character, a bell sounds and the cursor doesn't move.

f*x*	Moves the cursor right to character *x*
F*x*	Moves the cursor left to character *x*
t*x*	Moves the cursor right to the character just before character *x*
T*x*	Moves the cursor left to the character just after character *x*
;	Continues the search specified in the last command, going in the same direction
, (comma)	Continues the search specified in the last command, but goes in the opposite direction

On the following screen, if the cursor were at the beginning of the line and you entered the command **fA**, the cursor would move right to the first occurrence of the letter `A`. If you entered the command **tA**, the cursor would move to the space preceding `A`.

```
Go to the letter A on this line.
```

The **F** and **T** commands function similarly. If the cursor were at the end of the line on the previous screen and you entered the command **FA**, the cursor would move left to the A. If you entered **TA**, the cursor would move to the space following A.

The semicolon (;) and comma (,) commands continue a search. For example, if the cursor were at the beginning of the line on the previous screen and you typed **ft**, the cursor would move to the first t on the line. If you then typed a semicolon (;), the search would continue and the cursor would move to the next t on the line.

9.5.3 Positioning the Cursor by Lines

In addition to the **j** and **k** commands you have already used, the plus sign (+), minus sign (–), and Return commands move the cursor to other lines.

Moving Up a Line

The – command moves the cursor up a line, positioning it at the first non-blank character on the line. To move more than one line at a time, precede the – command with the number of lines you want to move. For example, to move the cursor up 13 lines, you would type this command:

13–

If some of these lines are above the text shown on the screen, the text scrolls up to display them. If there are fewer then 13 lines above the current line, a bell sounds and the cursor remains on the current line.

Moving Down a Line

Both the plus sign (+) and Return move the cursor down a line. You can precede these commands with the number of lines you want to move. For example, to move the cursor down nine lines, type one of these commands:

9+

9 Return

If some of the lines are below the current screen, the text will scroll down to display them. If there are fewer than nine lines below the screen, the cursor will not move.

9.5.4 Positioning the Cursor by Words

The **vi** editor considers a word to be a string of characters that can include letters, numbers, or underscores. The lowercase word-positioning commands (**w**, **b**, and **e**) consider punctuation to be a separate word. For example, the lowercase commands would interpret the following line as having 11 words.

```
A big, black dog ran in front of me!
```

The uppercase commands (**W**, **B**, and **E**) treat punctuation as part of the word. These commands would interpret the above line as having 9 words.

The following paragraphs describe the word-positioning commands.

w	Moves the cursor forward to the first character in the next word
W	Ignores all punctuation and moves the cursor forward to the next word

To illustrate the difference between these two commands, if the cursor were on the word `cliffs` in the following screen and you entered the command **w**, the cursor would move to the period following `cliffs`. If you entered the command **W**, the period would be ignored and the cursor would go to the word `An` instead.

```
The roar of the ocean echoed from the cliffs.  An
approaching storm darkened the sky.
```

You can precede the **w** and **W** commands with the number of words you want to move. The end of the line does not stop the cursor movement. Instead, the cursor wraps around and continues counting words from the beginning of the next line.

e Moves the cursor forward to the last character in the word (if the cursor is already on the last character of a word, it moves to the last character of the next word)

E Moves the cursor in the same manner as the **e** command, but treats punctuation as part of a word

Using the following screen as an example, if the cursor is on the first letter of `cliffs`, the **e** command will move it to `s`, the last letter in `cliffs`. If you issued the **E** command instead, the cursor would move to the period following `cliffs`.

```
The roar of the ocean echoed from the cliffs.  An
approaching storm darkened the sky.
```

You can precede the **e** and **E** commands with the number of words you want to move. Cursor movement does not stop at the end of the line. The cursor wraps around to the next line and continues counting words.

b Moves the cursor backward to the first character of the previous word

B Moves the cursor in the same manner as the **b** command, but treats punctuation as letters of a word

You can precede these commands with the number of words you want to move. The cursor does not stop at the beginning of a line, but moves to the end of the preceding line and continues counting backward.

9.5.5 Positioning the Cursor by Sentences

In **vi**, a sentence begins at the end of the previous sentence and ends in a period, exclamation mark, or question mark. If a sentence ends in the middle of a line, **vi** can recognize it as a sentence only if it is followed by two spaces. The following commands move the cursor by sentences:

(	Moves the cursor to the beginning of the current sentence
)	Moves the cursor to the beginning of the next sentence

You can precede these commands with a number to indicate how many sentences to move. For example, the command **2(** moves the cursor to the beginning of the previous sentence; **2)** moves the cursor ahead two sentences.

9.5.6 Positioning the Cursor by Paragraphs

The **vi** editor recognizes paragraphs if they are preceded and followed by a blank line. If you want to treat paragraphs as units, follow each paragraph with a blank line. The following commands move the cursor by paragraphs:

{	Moves the cursor to the beginning of the current paragraph
}	Moves the cursor to the beginning of the next paragraph

You can precede these commands with a number indicating how many paragraphs you want to move. For example, the command **3{** moves the cursor to the third paragraph above the current paragraph. **3}** moves the cursor to the third paragraph below the current paragraph.

9.5.7 Positioning the Cursor on the Screen

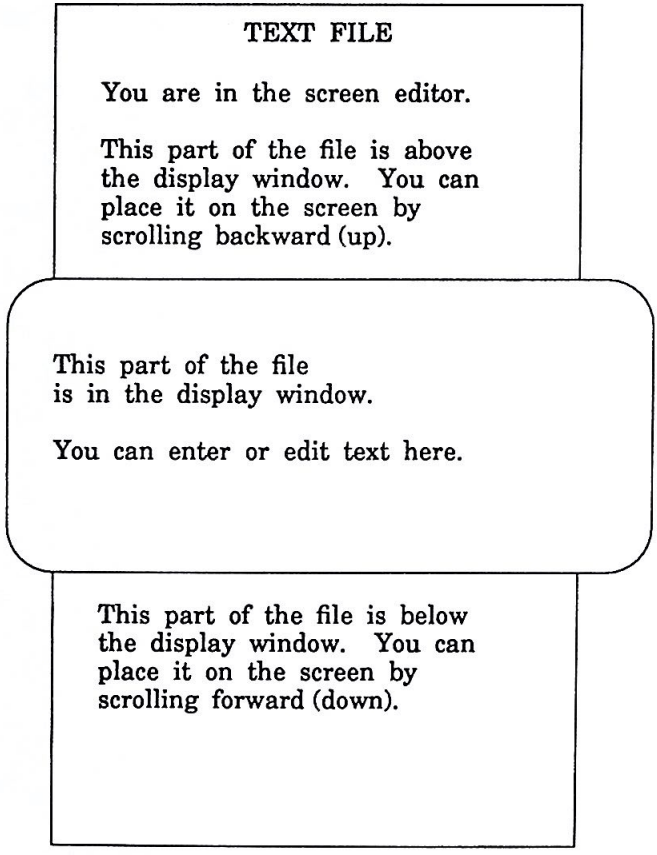
The **vi** editor provides three commands you can use to position the cursor on the screen. Be sure to type them in uppercase.

H	Moves the cursor to the first line on the screen
M	Moves the cursor to the middle line on the screen
L	Moves the cursor to the last line on the screen

9.5.8 Scrolling Text

The **vi** editor includes four commands you can use to scroll text on your screen. The Ctrl-F and Ctrl-D commands scroll text forward; the Ctrl-B and Ctrl-U commands scroll text backward.

The following illustration shows the movement that occurs when you scroll text on the screen.



NOTE

Not all terminals have text-scrolling capability. The type of terminal you have will determine whether you can take advantage of **vi**'*s scrolling feature.*

Scroll Forward a Full Screen

The Ctrl-F command scrolls one full window of text forward on your screen. To do this, **vi** clears the screen and then redraws the window. (The window is the area of your screen that displays text.) It moves the two lines that were at the bottom of the window to the top of the window. If there are not enough lines left in the file to fill the window, a tilde (~) appears to show that there are empty lines.

Scroll Forward a Half Screen

The Ctrl-D command scrolls a half window of text forward on your screen. The lines that were at the top of the screen disappear, allowing lines that were below the screen to be displayed. If there are not enough lines in the file to scroll forward a half window, a bell sounds.

Scroll Backward a Full Screen

The Ctrl-B command scrolls back a full screen of text. To do this, **vi** clears the screen and redraws the window around the text that had been above the current screen. If there are not enough lines above the current window to fill a new window, a bell sounds.

Scroll Backward a Half Screen

The Ctrl-U command scrolls back a half screen of text to reveal the lines just above the window. When the cursor reaches the top of the file, a bell sounds to notify you that the file can scroll no farther.

9.5.9 Line Numbers

The **vi** editor assigns a line number to each line of text in the buffer. This section tells how to display the number of the line containing the cursor and how to move the cursor to a specific line.

You can also set up **vi** to display all line numbers on the screen. For more information about this feature, refer to "Modifying Your **vi** Environment" at the end of this chapter.

Displaying Line Numbers

The **vi** command Ctrl-G and the line editor command **:.=** both display the number of the current line. However, Ctrl-G displays some additional information about the file. To use this command, position the cursor on the appropriate line and type Ctrl-G. A status notice then appears at the bottom of the screen giving you the following information:

- The name of the file
- The status of the file (new or modified)
- The number of the line containing the cursor
- The total number of lines in the buffer
- The percentage of the total lines in the buffer represented by the current line

The following screen shows a typical message produced by the Ctrl-G command.

```
This line is the 35th line of the buffer.
The cursor is on this line.
.
.
.
The last line of the buffer is line 116.

"file.name" [modified] line 36 of 116 --34%--
```

You can also use the following line editor command to display the number of the line containing the cursor:

:.= Return

The editor then displays the current line number at the bottom of the screen.

```
This line is the 35th line of the buffer.
The cursor is on this line.
.
.
.
The last line of the buffer is line 116.

36
```

Moving to a Specific Line

The **G** command moves the cursor to a specific line in your text. To use this command, precede **G** with the number of the line you want to move to. For example, to go to the first line in a file, you would enter **1G**. If the line you specify is not currently on the screen, the command clears the screen and then redraws the window around that line. If you do not specify a line number, the **G** command moves the cursor to the last line of the file.

You can also move the cursor to any line in your text by typing a colon (**:**) followed by the line number. For example, the following command moves the cursor to line 10:

:10 Return

When you want to indicate a specific line in a command, you can use a symbolic address in place of the actual line number. The symbolic addresses are as follows:

.	Represents the current line
$	Represents the last line in the file

,	Represents all lines in the file
;	Represents all lines from the current line to the last line

Other Information

"Using Symbols to Identify Lines" in Chapter 8

9.5.10 Searching for a Pattern of Characters

The slash (/) and question mark (?) commands search forward or backward in the buffer for the next occurrence of a specific character pattern. The commands appear on the screen as you type them and must be followed by Return. When you specify a search pattern, keep in mind that the commands can distinguish between upper- and lowercase characters.

The search commands are as follows:

/pattern	Searches forward for the specified *pattern*
?*pattern*	Searches backward for the specified *pattern*
n or /	Repeats the last search command
N or ?	Repeats the last search command, but searches in the opposite direction

The / command (invoked as */pattern*) searches forward in the buffer for the next occurrence of the characters in *pattern*. It then places the cursor on the first of those characters. For example, the following command finds the next occurrence of the words `Hello world` and places the cursor on the `H`.

/Hello world Return

The **?** command (invoked as **?***pattern*) searches backward in the buffer for the first occurrence of the characters in *pattern*. It then places the cursor on the first of those characters. For example, the following command finds the first occurrence (preceding your current position) of the words `data set design` and places the cursor on the `d` in `data`.

?data set design Return

These search commands do not wrap around the end of a line while searching for a pattern consisting of more than one word. For example, you might be searching for the words `Hello world`. If `Hello` is at the end of one line and `world` is at the beginning of the next line, the search command will not find that occurrence of `Hello World`.

However, the search commands do wrap around the end or the beginning of the buffer to continue a search. For example, if you are near the end of the buffer and the pattern for which you are searching (with the */pattern* command) is at the top of the buffer, the command will find the pattern.

You can use the **n** and **N** commands to continue a search. These commands do not appear on the screen. The **n** command continues the search in the same direction; the **N** command continues the search in the opposite direction. You don't have to press Return after entering these commands. Typing / Return has the same effect as **n**; typing **?** Return has the same effect as **N**.

The and **?** search commands do not allow you to use numbers to specify particular occurrences of a *pattern*. For example, you cannot request that a search go directly to the third occurrence of a *pattern*.

Using Special Characters in a Search Pattern

A search pattern (also called a regular expression) can include special characters that act as wildcards for matching characters. The special characters are as follows:

.	Matches one character
*	Matches zero or more occurrences of the preceding character
.*	Matches zero or more occurrences of any character following the period
^	Matches the beginning of the line
$	Matches the end of the line
&	Repeats the last replacement pattern
[...]	Matches the first occurrence of any character in the brackets
[^...]	Matches the first occurrence of any character not in the brackets

The section "Matching Patterns," located in Chapter 8, shows how to use these characters.

9.6 Undoing Commands

Before you begin working with the delete commands, you should be familiar with the **u** and **U** commands. These commands allow you to undo commands and return your file to its previous state.

u	Undo the last command you issued.
U	Restore the current line to its previous state.

If you accidentally issue a command that deletes or changes text, type **u** and the text will reappear on the screen. You must type **u** immediately after entering the incorrect command.

The **u** command can also undo a previous **u** command.

The **U** command cancels all changes made to the current line as long as you have not moved the cursor from it.

9.7 Deleting Text

The delete commands remove unwanted text and then redraw the affected part of the screen. The editor must be in command mode to use these commands.

There are two basic delete commands: **x** and **d**. The **x** command deletes characters. The **d** command, in combination with a cursor-movement command, deletes words, lines, sentences or paragraphs. (For descriptions of the cursor-movement commands, see the earlier sections on moving the cursor.)

You can also precede the **x** and **d** commands with a number to specify how many words, lines, sentences, or paragraphs you want to delete.

9.7.1 Deleting Characters

If you want to delete a character, move the cursor to that character and type **x**. The character will then disappear and the line will readjust to the change. To erase several characters in a row, either type **x** for each character or precede **x** with the number of characters you want to delete.

For example, you might want to delete the second occurrence of the word deep on the following screen.

```
Tomorrow the Loch Ness monster
shall slither forth from
the deep dark deep depths of the lake.
```

After moving the cursor to deep, enter the command **5x**. This command deletes five characters: the four letters in deep and the following space. The screen now looks like this:

```
Tomorrow the Loch Ness monster
shall slither forth from
the deep dark depths of the lake.
```

9.7.2 Deleting Words

The **dw** command deletes a word or part of a word. To use this command, move the cursor to the first character to be deleted and type **dw**. The character under the cursor and all subsequent characters in that word will be erased.

The **dw** command deletes one word or punctuation mark and the following spaces. To delete several words or punctuation marks at once, specify a number before the command. For example, to delete the three words and two commas deep, deep, dark on the following screen, you would move the cursor to the first occurrence of **deep** and then type **5dw**.

```
the deep, deep, dark depths of the lake
```

The line will now look like this:

```
the depths of the lake
```

9.7.3 Deleting Lines

The following commands delete lines:

dd	Deletes an entire line
d0 (zero)	Deletes from the beginning of the line to the character before the cursor

To delete an entire line, type **dd**. To delete multiple lines, specify a number before the command. For example, typing **10dd** will erase 10 lines. If you delete more than a few lines, **vi** will print a message such as the following on the bottom of the screen:

```
10 lines deleted
```

In this example, if there are fewer than 10 lines below the current line, a bell sounds and no lines are deleted.

The **d0** command deletes all characters from the beginning of the line through the character preceding the cursor.

9.7.4 Deleting the Rest of the Line

To delete characters starting from a specific character and going to the end of the line, first move the cursor to that character, then type either of these commands:

D

d$

These commands do not allow you to specify how many lines to delete. They apply only to the current line.

9.7.5 Deleting Sentences

The following commands delete sentences:

d(	Deletes all characters from the beginning of the sentence to the cursor position
d)	Deletes all characters from the cursor position to the end of the sentence

You can precede these commands with a number to indicate how many sentences you want to delete. For example, the command **2(** deletes the current sentence and the previous sentence; **2)** deletes the remainder of the current sentence and the following sentence.

9.7.6 Deleting Paragraphs

The following commands delete paragraphs:

d{	Deletes all characters from the beginning of the current paragraph to the cursor position
d}	Deletes all characters from the cursor position to the end of the current paragraph

You can precede these commands with a number to specify how many paragraphs you want to delete. For example, the command **2d{** deletes the preceding paragraph and all characters up to the cursor position in the current paragraph. The command **2d}** deletes the remainder of the current paragraph (starting at the cursor position) and the following paragraph.

9.7.7 Deleting the Rest of the Buffer

You can use the line editor command **:d** in combination with the symbols for the current and last lines to delete all text following the current line. (The period (.) represents the current line; the dollar sign ($) represents the last line.) Because this is an editor command, you must press Return to activate it.

:.,$d Return

9.8 Modifying Text

This section describes several commands that allow you to delete and create text simultaneously.

9.8.1 Replacing Text

r	Replaces the character the cursor is positioned on
R	Replaces text from the cursor position to the end of the line, then adds text until you press Esc

The **r** command replaces the current character with the next character you type. For example, suppose you want to change the word `ants` to `acts`.

```
The circus has many ants.
```

Move the cursor to the `n` in `ants` and type **rc**. The line now looks like this:

```
The circus has many acts.
```

If you precede the **r** command with a number, it will replace that number of characters with the letter you specify. You don't need to press Esc after using this command; it automatically terminates after replacing the specified number of characters. Using the line from the preceding example, you could change `many` to `7777` by placing the cursor under the `m` of `many` and then typing this command:

4r7

On the following screen, the **r** command changes the four letters in `many` to four occurrences of the number 7.

```
The circus has 7777 acts.
```

The **R** command replaces all characters on a line starting from the cursor position. When you type **R**, the editor goes into input mode, allowing you to write over the remaining characters on the line. You can enter as much text as you want; the editor stays in input mode until you press Esc.

If you will be entering fewer characters than currently exist on the line, use the **C** command (described later in this chapter). The **R** command will leave any extra characters on the line after you press Esc.

9.8.2 Substituting Text

The substitute commands replace characters, but allow you to continue inserting text until you press Esc. The substitute commands are as follows:

s	Deletes the character shown by the cursor, then appends text
S	Replaces all characters in the line

You can precede the **s** command with a number to indicate how many characters to substitute. A `$` sign then overwrites the last character to be substituted. The characters are not erased from the screen until you type over them or leave input mode by pressing Esc.

For example, you might want to substitute the word `million` for the word `hundred` on the following screen. Place the cursor on the `h` of `hundred` and type **7s**. A dollar sign (`$`) appears in place of the last letter in `hundred`, indicating that all letters up to that point will be substituted.

```
My salary is one hundre$ dollars.
```

Then type **million**. The line now looks like this:

```
My salary is one million dollars.
```

You can substitute any number of characters. For example, you could have replaced `hundred` with `thousand`, which contains a different number of characters. **vi** will continue to add text until you press Esc to terminate the command.

The **S** command replaces an entire line. You can enter the command from any position on the line. When you type **S**, it first deletes all characters in the line. It then moves the cursor to the beginning of the line and puts **vi** in input mode. You can enter as much text as you want; **vi** remains in input mode until you press Esc.

9.8.3 Changing Text

The change commands replace words, lines, sentences, or paragraphs. These commands are similar to the delete commands. However, they put **vi** into input mode and allow you to enter text until you press Esc.

You can combine the basic change command (**c**) with any of the cursor movement commands that affect words, sentences, or paragraphs. (For a description of these commands, see the earlier sections on moving the cursor.)

You can also precede the change commands with a number that specifies how many words, lines, sentences, or paragraphs you want to change.

Changing Words

The **cw** command replaces a word or the remaining characters in a word with new text. When you type this command, a dollar sign (`$`) marks the last character that will be replaced. For example, you might want to change `Tuesday` to `Wednesday` on the following screen. After moving the cursor to `T` and entering the **cw** command, `$` replaces the letter `y` in Tuesday.

```
They are now due to arrive on Tuesda$ afternoon.
```

Then type `Wednesday` and press Esc. The line now looks like this:

```
They are now due to arrive on Wednesday afternoon.
```

Notice that the new word (`Wednesday`) has more letters than the word it replaced (`Tuesday`). The **cw** command places **vi** in input mode, allowing you to enter as much text as you want.

Changing Lines

You can use the following commands to change the text on a line:

cc	Replace all characters on the line
C	Replace all characters from the cursor to the end of the line

The **cc** command deletes all characters on the current line, then puts **vi** in input mode. You can then enter as much text as you want, including new lines. When you finish entering text, press Esc to return to command mode. You can invoke **cc** from any location on the line.

The **C** command starts replacing characters at the cursor position and marks the end of the line with a `$` sign. It puts **vi** in input mode and waits for you to type new characters over the old text. In this example, we invoked the **C** command when the cursor was on the `p` in `paragraph`.

```
This is line 1
This is paragraph number $
This is line 3
This is line 4
```

We then changed the rest of the line, as shown below.

```
This is line 1
This is line 2
This is line 3
This is line 4
```

Changing Sentences

The following commands change sentences:

- **c(** Replaces all characters from the beginning of the sentence to the character before the cursor position
- **c)** Replaces all characters from the cursor position to the end of the sentence

Changing Paragraphs

These commands change paragraphs:

- **c{** Replaces all characters from the beginning of the paragraph to the character before the cursor position
- **c}** Replaces all characters from the cursor position to the end of the paragraph

9.8.4 Making Global Changes

The global commands locate or replace a specific pattern of characters either on the current line, in a section of the file, or in the entire file. The **:s** command replaces the first occurrence of the pattern in a line. The **:g** command locates or changes every occurrence of the pattern. You must press Return to activate these commands. The global commands are as follows:

:g/*pattern* **/***command*

This command locates each line containing the specified pattern, then executes the specified **ex** command. For example, the following command will display each line containing the word `helix`:

:g/helix/p Return

:s/*pattern/new_words*

This command applies only to the current line. It replaces the first occurrence of the specified *pattern* with the *new_words*. For example, the following command will substitute the words `double helix` for the first occurrence of the pattern `helix`.

:s/helix/double helix Return

:s/*pattern/new_words***/g**

This command is similar to the previous command. However, the **/g** at the end of the line tells the editor to change every occurrence of *pattern* on the current line.

You can precede the **s** portion of these commands with a line number or range of line numbers that specify where the substitution should take place. For example, the following command would apply to lines 1 through 10:

:1,10s/helix/double helix/g Return

Similarly, the following command would apply from line 15 through the current line (the **.** represents the current line):

:15,.s/helix/double helix/g Return

:g/*pattern***/s//***new_words***/g**

This command applies to the entire file. It replaces each occurrence of the specified *pattern* with the *new_words*. For example, the following command will change every occurrence of the word `helix` to `double helix`.

:g/helix/s//double helix/g Return

The *pattern* used with these commands can include the special characters described earlier in this chapter.

The section "Substituting Text: **s**, **g**," located in Chapter 8, contains additional examples of global commands.

9.9 Cutting and Pasting Text

The **vi** editor provides commands to cut and paste text. You can use the delete commands described earlier in this chapter to cut out a piece of text, or you can use the yank command described later in this section to copy a piece of text. After deleting or yanking the text, you can use the **p** or **P** command to place (or paste) the text in another location in the file.

9.9.1 Pasting Text

The **p** and **P** commands allow you to paste text anywhere in your file. These commands can paste deleted text or a copy of existing text.

p	Places the contents of the temporary buffer after the cursor
P	Places the contents of the temporary buffer before the cursor

When you use the delete or yank commands, **vi** places the deleted or yanked characters into a temporary buffer. You can then use the **p** or **P** command to paste the contents of the buffer into another location in your file. Because the temporary buffer stores the results of the last command only, invoke the **p** or **P** command immediately after using a delete or yank command.

For example, if you used the **D** command to remove part of a sentence, you could place the deleted words in the middle of another line. To do this, position the cursor at the appropriate location in the line and type **p**. The editor then places the partial line after the cursor. If you wanted to insert the words before the cursor, you would use the **P** command instead.

9.9.2 Fixing Transposed Letters

If you need to fix transposed letters, simply combine the **x** and the **p** commands to form the **xp** command. **x** deletes the first character; **p** places it after the next character.

For example, notice the error in the following line.

```
A line of tetx
```

To correct this error, place the cursor on the second `t` in `tx` and then type **xp**. The line now looks like this:

```
A line of text
```

9.9.3 Copying Text

The yank command copies one or more lines of text into a temporary buffer. The lines will also remain in their original position in the file. If you want the same text to appear in more than one place, the yank command provides a convenient way to avoid typing the text several times.

The basic yank command is **y**. You can combine this command with the commands that move the cursor by words, sentences, or paragraphs. (For a description of these cursor-movement commands, see the earlier sections on moving the cursor.)

yy	Yanks the current line
yw	Yanks the current word
y)	Yanks the current sentence
y{	Yanks the current paragraph

For example, to yank a sentence, you would move the cursor to that sentence and type **y)**; to yank a paragraph, you would move the cursor to that paragraph, then type **y}**. If you yank more than a few lines, a message stating the number of lines yanked appears at the bottom of the screen.

You can also precede these commands with a number to specify how many words, lines, sentences, or paragraphs should be yanked. For example, the command **5yw** yanks five words, the command **4yy** yanks four lines, and the command **3y}** yanks three paragraphs.

The editor places the yanked text into the temporary buffer. Next, move the cursor to the new location for the text, then invoke either **p** or **P** to paste the text from the temporary buffer into the new location.

9.9.4 Using Registers to Copy or Move Text

Moving or copying several sections of text to a different part of a file can be time-consuming. **vi** provides a shortcut for this: named registers that can store text until you are ready to place it in a new location. You can either yank or delete the text you want to store. The extracted text remains in the specified register until you either end the editing session, or yank or delete another section of text to that register.

There are 26 named registers; their names are the letters a through z. To use these registers, first type a double quotation mark ("), type the name of the register, then type the appropriate yank or delete command. For example, the following command yanks the current line into register c:

```
"cyy
```

You can precede the command with the number of lines, sentences, or paragraphs to be yanked or deleted. For example, the following command places three sentences into register a:

```
3"ay(
```

To paste a copy of the register's contents into your text, move the cursor to the appropriate location, then type " followed by the name of the register and the **p** command. For example, to paste the contents of register a, you would enter this command:

```
"ap
```

Similarly, the command "**cp** will paste the contents of register c.

9.10 Miscellaneous Commands

This section describes some special commands you might find useful:

.	Repeats the last command
J	Joins two lines together
Ctrl-L	Clears the screen and then redraws it
~	Changes lowercase to uppercase and vice versa
:r	Reads another file into the buffer
:sh and **:!**	Returns to the shell

It also tells how to use markers to indicate a specific point in your text and how to run the **spell** command to check the spelling in a file.

9.10.1 Repeating the Last Command

The period (.) command repeats the last command you issued to create, delete, or change text. It is often used with the search command.

For example, suppose you forget to capitalize the `S` in `United States`. However, you do not want to capitalize the `s` in `chemical states`. One way to correct this problem is to search for the word states. The first time you find it in the expression `United states`, you can change the `s` to `S`. Then continue your search. When you find another occurrence, you can simply type a period; **vi** will remember your last command and repeat the substitution of `s` for `S`.

If you try to add a period to the end of a sentence while **vi** is in command mode, the text change resulting from your last command will appear on the screen.

9.10.2 Joining Two Lines

The **J** command joins lines. To use this command, place the cursor on the current line and type **J**. **vi** then joins the current line and the following line.

For example, you might have the following two lines of text:

```
Dear Mr.
Smith:
```

To join these two lines into one, place the cursor on any character in the first line and type **J**. The command will immediately join the two lines on your screen:

```
Dear Mr. Smith:
```

Notice that **vi** automatically inserts a space at the point where the lines join (between `Mr.` and `Smith` in our example).

9.10.3 Clearing and Redrawing the Window

If another user invokes the **write** command to send you a message while you are editing with **vi**, the message will appear on your screen over part of the text you are editing. To restore your text, place **vi** in command mode. Then type Ctrl-L. **vi** will erase the message and redraw the window exactly as it appeared before the message arrived.

9.10.4 Changing Letter Case

The tilde (~) command changes a lowercase letter to uppercase, or vice versa. To use this command, move the cursor to the letter to be changed, then type ~. For example, to change the letter a to A, press ~. You can change several letters by typing ~ several times, but you cannot precede the command with a number to change several letters with one command.

9.10.5 Adding a File to the Buffer

To add another file to the text you are editing, use the **:r** (read) command. This command inserts the new file below the current line of text. For example, to read the contents of a file called *data* into your current file, place the cursor on the line above the place where you want to add the *data* file. Then type this command:

:r data Return

Instead of moving the cursor, you can specify the appropriate line number in the command. For example, to insert the *data* file below line 56 of the buffer, you would type this command:

:56r data Return

9.10.6 Returning to the Shell

When you are using **vi**, the contents of the buffer fill your screen, making it impossible to issue shell commands. **vi** offers two methods of escaping the editor temporarily so that you can issue shell commands (and even edit other files) without having to write the buffer and quit. These commands are **:!** and **:sh**.

The **:!** command allows you to escape the editor and run a single shell command. While **vi** is in command mode, type **:!**. (These characters appear at the bottom of your screen.) Then type a shell command and press Return. The shell will run your command, give you output, and print the message `[Hit return to continue]` when the command completes. When you press Return, **vi** will refresh the screen and the cursor will reappear exactly where you left it.

The **:sh** command also allows you to execute shell commands, but behaves differently on the screen. While **vi** is in command mode, type **:sh** and press Return. A shell command prompt then appears on the next line. Type your commands after the prompt as you would normally do while working in the shell. When you are ready to return to **vi**, type either Ctrl-D or **exit**; **vi** will refresh your screen and the cursor will appear where you left it.

You can even change directories while you are temporarily in the shell. When you type **exit** or Ctrl-D, the cursor will appear where it was when you left **vi**. If you changed directories while you were in the shell, you will be back in your original directory when you return to **vi**.

9.10.7 Using Markers

When you are using **vi**, you can set markers anywhere in your text. You can use up to 26 markers; the markers are named a through z. The markers you set stay in effect until you either reset them or finish editing the file.

To set a marker, type **m** followed by the name of the marker. For example, to set marker r, you would type **mr**.

When you want to return to a marker, type ' followed by the name of the marker. To return to marker r, you would type **'r**.

You can frequently use markers in place of line numbers. For example, you can include them in a substitution command. If you wanted to change the word `work` to `exercise` starting at marker a and going to marker b, you would type this command:

:'a,'bs/work/exercise/g Return

You can also use markers with the delete command. For example, the command **d'f** deletes all text from the current line to marker f.

9.10.8 Locating Spelling Errors

The **spell** shell command checks a file for spelling errors. The operating system maintains a dictionary that **spell** uses to check each word in your file. It then lists, in alphabetical order, all words from your file that it cannot find in the dictionary.

To use this command, you must be at the shell prompt. Type **spell** followed by one or more filenames. For example, to check the spelling in the *memos* file, you would type this command:

$ **spell memos** Return

If you specify more than one filename, **spell** displays all misspelled words in a single list.

If you are checking spelling in a large file, you can redirect the output from **spell** to a file, rather than printing the output on your screen. For example, the following command redirects the output from the *memos* file to a file named *memos.spell*.

$ **spell memos > memos.spell** Return

Other Information

spell(1)

9.11 Modifying Your vi Environment

The **vi** editor uses several parameters, or variables, to determine how it should operate on your terminal. For example, there are parameters that indicate the type of terminal you are using and that specify the number of lines to display on your screen. You can use the **set** command to display all of the parameters that affect **vi** or only those options that have been reset from their default values.

You can also use the **set** command to change the value of an option. You can reset an option for the current **vi** session only or you can reset it permanently.

9.11.1 Displaying Parameters

To display all of the **vi** parameters, enter the following command while you are working in **vi**:

:set all Return

A list of parameters then appears on your screen. Some parameters take a value, such as `window=24`. Other parameters are either on or off. If a parameter is off, it is preceded by `no`, such as `noreadonly`. Otherwise the parameter is on.

To view only the parameters that have been reset from their default values, type this command while you are working in **vi**:

:set Return

A list of the parameters that have been reset on your terminal will then appear at the bottom of your screen.

9.11.2 Changing Parameters

You can also use the **set** command to change any of the **vi** parameters. For most parameters, you can just use the default values. However, the remainder of this chapter lists several parameters you might want to change.

To set a parameter for the current **vi** session only, invoke **:set** followed by the parameter. If the parameter is either on or off, type the parameter name to turn it on; type the parameter name preceded by `no` to turn it off. For example, the following command turns on the parameter to display line numbers. (You must be working in **vi** when you invoke this command.)

:set number Return

If the parameter takes a value, type the parameter name or abbreviation. Then type = followed by the value. For example, the following command sets the wrapmargin parameter for a right margin of 5 characters, at which point **vi** will insert an automatic return.

:set wm=5 Return

If you want to change some parameters permanently, create a file called *.exrc* in your home directory. Then set the parameters in this file. You can set all of the parameters on a single line. For example, to set the wrapmargin and line numbers parameters, you would enter the following line in *.exrc*. The parameters will then be in effect each time you invoke **vi**.

```
set wm=5 number
```

Setting an Automatic Return

The **wrapmargin** parameter (abbreviated as **wm**) causes **vi** to insert an automatic return whenever the cursor reaches a specific character position. The default value for this parameter is `wrapmargin=0`, which means **vi** does not insert an automatic return.

If you want to use this parameter, you must specify a character position for the automatic return. To do this, indicate the number of spaces from that character position to the right margin of the screen. For example, if you add the following line to your *.exrc* file, **vi** wll insert an automatic return whenever the cursor is 10 characters from the right margin.

```
set wm=10
```

Displaying Line Numbers

The **number** parameter determines if line numbers are displayed on your screen. Because the default value is `nonumber`, **vi** doesn't normally display line numbers. If you choose to display line numbers, they will not be stored with the file and they won't appear on printouts. They appear only on the screen. To turn on this option, use the command **:set number** or add `set number` to your *.exrc* file.

Displaying Input Mode

The **showmode** parameter displays a message such as `INSERT`, `APPEND`, or `CHANGE` whenever you enter a command that puts **vi** in input mode. The default value for this parameter is `noshowmode`. To turn on this option, use the command **:set showmode** or add `set showmode` to your *.exrc* file.

Ignoring Case in Searches

The **ignorecase** parameter determines if **vi** distinguishes upper- and lowercase letters when performing search commands. The default value is `noignorecase`, causing **vi** to make this distinction. To turn on this option, use the command **:set ignorecase** or add `set ignorecase` to your *.exrc* file.

Displaying Tabs and Marking Ends of Lines

The **list** parameter determines how tabs appear on the screen and how the end of a line is marked. The default value, `nolist`, displays tabs as white space and does not mark the end of each line. To turn on this option, use the command **:set list** or add `set list` to your *.exrc* file. When the **list** parameter is on, **vi** displays tabs as `^|` and marks the end of each line with a `$`.

Setting Automatic Indentation

The **autoindent** parameter (abbreviated as **ai**) provides a set of indentations for programs or other material requiring tabs. (The **shiftwidth** parameter specifies the number of characters to indent.) The default value for this parameter is `noautoindent`. To turn on this option, use the command **:set ai** or add `set ai` to your *.exrc* file.

When **autoindent** is on and **vi** is in input mode, you can use Ctrl-T to move the cursor to the next indentation position to the right. Ctrl-D moves the cursor to the previous indentation position. These keys work in this manner only before you enter text on the line. Pressing Return moves the cursor to the next line and aligns it with the first character of the previous line.

Setting Shiftwidth

The **shiftwidth** parameter determines the spacing of the indentations used by the **autoindent** option. The default value is `shiftwidth=8`, however you can change it to another value if you prefer.

Other Information

ex(1)

Chapter 10
Communications

10.1 Introduction 10-1

10.2 Reading System-Wide News 10-1
Posting News Items 10-2

10.3 Sending Messages to Another User's Terminal 10-2

10.4 Using mail 10-4
Sending Mail 10-4
Managing Incoming Mail 10-5
Reading Messages 10-5
Displaying the Next Message 10-6
Deleting Messages 10-7
Saving Messages 10-7
Redisplaying a Message 10-7
Mailing a Message to Another User 10-8
Returning to the Shell 10-8
Quitting **mail** 10-8
Mailing Files 10-9

10.5 Using mailx 10-10
mailx Files 10-10
Writing Mail 10-10
Specifying a Subject on the Command Line 10-12
Working in Input Mode 10-12
Canceling Mail 10-12
Editing Mail 10-13
Previewing a Message 10-13
Incorporating Existing Text 10-14
Changing the Message Header 10-16
Adding Your Signature 10-18
Returning to the Shell 10-18
Executing Command Mode Commands 10-19
Keeping Copies of Your Messages 10-19
Managing Mail: Command Mode 10-20
Working in Command Mode 10-21
Reading Mail 10-22

Exiting **mailx** 10-23
Displaying Header Information 10-24
Reading the First Lines of a Message 10-25
Reading Other Mail Files 10-25
Deleting Mail 10-26
Saving Mail 10-27
Copying Messages 10-28
Holding Messages in Your Mailbox 10-28
Replying to Mail 10-29
Answering Mail and Saving Your Reply 10-30
Editing Messages 10-30
Piping a Message Through a Command 10-30
Returning to the Shell 10-31
Mailing Files 10-31
Setting Up Your **mailx** Environment 10-32
Displaying Variables 10-32
Modifying Variables 10-33
Modifying Variables for the Current Login Session 10-33
Modifying Variables Permanently 10-34
mailx Environment Variables 10-35
Creating Aliases 10-40
Adding Other Commands to a ***.mailrc*** File 10-40

10.6 Using UUCP 10-41
UUCP Communications 10-41
Communicating with Users on Other Systems 10-41
Finding the Name of Another Site 10-41
Determining if Your Site and a Remote Site Can Communicate . 10-42
Sending Mail via UUCP 10-43
Transferring Files to Remote Systems 10-43
Transferring Files to a Specific Directory: **uucp** 10-44
uucp Options 10-46
Transferring Files to a Public Directory: **uuto** 10-47
Checking Job Status: **uustat** 10-48
Receiving Files: **uupick** 10-50
Printing Files 10-51
Moving Files 10-51
Deleting Files 10-52
Skipping a File 10-52
Exiting **uupick** 10-52
Executing Commands on a Remote System: **uux** 10-53
Connecting a Remote Terminal: **ct** 10-54
Connecting a Remote System: **cu** 10-56

cu Options 10-57
Executing Commands 10-58

Figures

10-1 Sample *.mailrc* file 10-34

Chapter 10
Communications

10.1 Introduction

The operating system includes many commands that enable you to communicate with other users. This chapter tells how to use these commands to perform the following tasks:

- Read and create system-wide news items.
- Write messages to another user's terminal.
- Send and receive electronic mail messages.
- Transfer files between machines.
- Execute commands on another machine.

10.2 Reading System-Wide News

The **news** feature allows system-wide posting of news items such as meeting notices or other announcements. If you want to read the news items, issue the **news** command, as shown below. The system will then display all news items posted since the last time you read news.

```
$ news Return
```

To stop printing a particular news item, press Delete. If you want to exit from **news**, press Delete twice.

If you don't want to read all of the news items, you can use the following options to determine specific items to read:

-n Lists only the names of the news items. To view particular news items, type **news** followed by the names of those items.

-s Lists the number of news items available.

If you don't want to read any of the news items, use the following command to print the news items on the null device, which in effect deletes the items.

```
$ news > /dev/null Return
```

10.2.1 Posting News Items

When you want to post a news item, use an editor such as **vi** to create a file in the */usr/news* directory. For example, the following command creates a news item named *survey*:

```
$ vi /usr/news/survey Return
```

After typing the text of your news item, you need to change permissions on the file. To allow everyone on your system to read your news item, the file must have read permission for all users. You can use either of the following commands to assign this permission:

```
$ chmod 444 /usr/news/survey Return

$ chmod a+r /usr/news/survey Return
```

Other Information

news(1)

10.3 Sending Messages to Another User's Terminal

The **write** command allows you to print a message on another user's terminal. You can also use **write** to carry out a conversation with another user over your terminals. To use this command, type **write** followed by the user's login name, press Return, and then type your message. For example, you could send this message to user *jmrs*:

```
$ write jmrs Return
Do you want to go to lunch? Return
```

The system then establishes a connection to the user's terminal and sends your message. At this point, the recipient of the message can write back. For example, *jmrs* might send this message:

```
$ write starship  Return
see you at noon  Return
```

When the conversation is over, both users must type Ctrl-D at the beginning of a line to return to the shell.

You might want to end your message with a signal (such as **(o)** for over) so the recipient will know when to reply. When you want to terminate the conversation, send a signal such as **(oo)** (for over and out).

The **mesg** command allows you to specify whether other users can use the **write** command to send messages to your terminal. This command has two values: **y** or **n**. The **y** value allows users to send messages to your terminal; the **n** value does not allow messages to appear. The default value for this command is **y**.

To determine the current value of **mesg** on your terminal, type **mesg** and press Return. If **mesg** is enabled, this message will appear:

```
$ mesg  Return
is y
```

If **mesg** is disabled, the message `is n` appears.

Many users choose to disable the **write** command, as it can interrupt work and make the screen display difficult to read. If you want to disable this feature for the current login session, type the following command:

```
$ mesg n  Return
```

If you want to disable **write** permanently, add this line to your *.profile* (for Bourne and Korn shell users) or *.login* (for C shell users):

```
mesg n
```

If you have disabled the **write** command and then want to enable it, remove the `mesg n` line from your *.profile* or *.login*, or type this command:

```
$ mesg y Return
```

Other Information

mesg(1), **write**(1)

10.4 Using mail

The **mail** command allows you to send and receive electronic mail. Although **mail** is not as sophisticated as the **mailx** program described later in this chapter, you can use it to send messages to other users and to read, save, or delete the messages you receive.

10.4.1 Sending Mail

To send a mail message, type **mail** followed by the recipient's login name and then press Return. For example, to send mail to user *helen*, you would type this command:

```
$ mail helen Return
```

Then type the text of your message. When the cursor reaches the right side of your screen, press Return to advance to the next line. You can type as many lines as you want. When you finish your message, type a period (.) or a Ctrl-D at the beginning of a new line to send the message.

The following example shows how this procedure appears on your screen.

```
$ mail helen  Return
My meeting with Smith is at 3:00.  Return
Could we meet in the morning?  Return
Fritz  Return
.  Return
$
```

You can send a message to several people by including their login names on the **mail** command line:

```
$ mail wildcat puma lynx dave Return
```

If you are typing a message and then decide you don't want to send it, press your interrupt key (typically Delete) to cancel the message. The shell prompt then appears. The **mail** program will save the text of your message in a file named *dead.letter* in your home directory.

If you want to try out the **mail** program, use the above procedure to send several messages to yourself. You can then use the commands described in the next section to read, delete, or save your messages.

10.4.2 Managing Incoming Mail

When other users send you messages, the **mail** program holds the messages in your system *mailbox*. This is a file, located in the */usr/mail* directory, that has your login name. For example, if you are login *puma*, your mailbox will be */usr/mail/puma*. The **mail** program provides a system mailbox for each user on your system.

If mail is waiting in your system mailbox when you log in, or if someone sends you mail while you are logged in, the following message prints on your screen:

```
you have mail
```

You can use the **mail** commands described in the remainder of this section to read, save, or delete your mail.

Reading Messages

To read the messages in your mailbox, type the **mail** command without any arguments:

```
$ mail Return
```

The last message you received then appears on the screen.

If you want to display your messages in chronological order instead, include the **-r** option on the command line. The oldest message in your mailbox will then display first.

```
$ mail -r Return
```

A typical **mail** message display looks like this:

```
$ mail Return
From tommy Wed May 21 15:33 CST 1986
Bob,
Looks like the meeting has been canceled.
Do you still want the material for the technical review?
Tom

?
```

The first line, called the header, contains the login name of the sender and the date and time the message was mailed. The lines after the header make up the text of the message. The question mark (?) is the prompt for the **mail** program.

If a long message is printing on your screen, it might scroll faster than you can read it. To stop the scrolling, type Ctrl-S. This will freeze the screen, giving you a chance to read the message. When you are ready to continue, type Ctrl-Q and scrolling will resume.

If you don't want to read an entire message, press your interrupt key (typically Delete). The **mail** program then stops the display and prints the **mail** prompt (?).

When a message finishes printing, the **mail** prompt (?) appears on your screen. You have many options at this point. You can leave the current message in your mailbox while you read the next message, you can delete the current message, or you can save the current message for future reference. For a list of available options, type a question mark at the **mail** prompt (?).

Displaying the Next Message

To see the next message in your mailbox, either press Return or type a plus sign (+) at the **mail** prompt (?).

```
? Return
```

```
? + Return
```

The current message remains in your mailbox and the next message appears. If you have read all of your messages, the shell prompt appears.

Deleting Messages

To delete the current message, type **d** at the **mail** prompt (?). (The message is not actually deleted from your mailbox until you exit **mail**.)

? **d** Return

After you execute this command, the next message in your mailbox appears. If there are no more messages, the **mail** prompt (?) appears.

Saving Messages

To save a message for later reference, type **s** at the **mail** prompt (?).

? **s** Return

The next message in your mailbox then appears.

By default, the **s** command saves the message in a file called *mbox* in your home directory. To save the message in another file, type the name of that file after the **s** command. For example, to save a message in a file called *calendar* (in your current directory), you would enter this command:

? **s calendar** Return

If *calendar* is an existing file, the **mail** program appends the message to it. If there is no file by that name, the **mail** program creates the file and stores your message in it.

If you want to save a message in a directory other than the current directory, specify a full or relative pathname for the file. For example, to save a message in a file called *memo* in your *project1* subdirectory, you would type this command:

? **s project1/memo** Return

Redisplaying a Message

To display a message again, type either **p** or a minus sign (–) at the **mail** prompt (?). The current message then reappears on the screen.

? **p** Return

? **–** Return

Mailing a Message to Another User

You can use the **m** command to mail the current message to other users. At the **mail** prompt (?), type **m** followed by the appropriate login names. The following command mails the current message to users *marie* and *puma*.

? **m marie puma** Return

Returning to the Shell

The exclamation (!) command allows you to leave **mail**, execute a shell command, and then return to the same place in **mail**. We used this command on the following screen to return to the shell and execute the **date** command. When the command completes, an exclamation point (!) appears on the screen, followed by the **mail** prompt (?).

```
? !date  Return
Mon Jul 18 11:24:31 PDT 1988
!
?
```

Quitting mail

There are two commands that allow you to leave the **mail** program: **q** and **x**. When you type **q** at the **mail** prompt (?), the messages you deleted are removed from your mailbox. Unread messages remain in your mailbox until the next time you use the **mail** program.

? **q** Return

To leave **mail** without changing the status of your messages, type **x** at the **mail** prompt (?). All of your messages will remain in your mailbox, including the messages you deleted or saved.

? **x** Return

10.4.3 Mailing Files

Occasionally you might want to mail a file to another user. To do this, use the name of the file as the input for the **mail** command.

mail *login* **<** *filename*

login is the recipient's login name; *filename* is the name of the file you want to send. For example, to send the *agenda* file to login *sarah*, you would type this command:

$ **mail sarah < agenda** Return

When *sarah* issues the **mail** command to read her messages, she will receive *agenda*.

To send the same file to several users, use the same command line format, separating the logins by spaces.

$ **mail sarah tommy wildcat < agenda** Return

Other Information

mail(1)

10.5 Using mailx

The **mailx** program is an enhanced version of the **mail** program. It contains many additional options for sending and reading mail. For example, you can use a text editor to create your messages.

mailx has two operating modes: input mode and command mode. When **mailx** is in input mode, you can create and send messages. When **mailx** is in command mode, you can read incoming mail, save, edit, or delete messages, forward messages to someone else, and respond to the person who originated a message. A different set of commands is available in each mode.

You can also use **mailx** environment variables to tailor **mailx** to suit your individual needs. The section "Setting Up Your mailx Environment," later in this chapter, tells how to do this.

10.5.1 mailx Files

The **mailx** program maintains a file, called a *mailbox*, for each user on the system. These files are stored in the */usr/mail* directory and take the user's login name. For example, if you are user *wildcat*, your mailbox will be */usr/mail/wildcat*. When other users send messages to you, the **mailx** program stores them in your mailbox.

When you issue the **mailx** command, you can read, save, or delete the messages in your mailbox. When you exit **mailx**, the **mailx** program removes deleted messages from your mailbox and transfers saved messages to the files you specified. It also transfers the messages you read, but did not explicitly save or delete, to a file named *mbox* in your home directory. Unread messages remain in your mailbox.

10.5.2 Writing Mail

To send a message to another user, type **mailx** followed by the user's login name. **mailx** then goes into input mode and prompts you for the subject of the message, as shown on the following screen.

```
$ mailx wildcat [Return]
Subject:
```

The subject line is optional. To include this line in your message, type a description of the message and press [Return]. To skip the subject line, simply press [Return].

When the cursor moves to the next line, you can type your message. Be sure to press [Return] when the cursor reaches the right side of the screen. This will break up the lines of your message, making it easier to read.

When you finish your message, type a tilde followed by a period (~.) at the beginning of a line. The message EOT (which stands for end-of-text) then appears, followed by the shell prompt. The following screen shows a sample message.

```
$ mailx wildcat [Return]
Subject: meeting time [Return]
The cats are meeting beside the creek [Return]
at 9:00 AM tomorrow. [Return]
Bob [Return]
~.  [Return]
EOT
$
```

To send mail to several users, type their login names on the command line, leaving spaces between the names. For example, the following command sends a message to users *tommy*, *wildcat*, and *puma*.

```
$ mailx tommy wildcat puma [Return]
```

You can also send mail to an *alias*, which is a word associated with specific login names. For example, the alias *mktg* could represent users *tom*, *patty*, and *trg*. When you send mail to an alias, a copy of the message goes to each user included in the alias. In this example, when you send mail to the *mktg*

alias, it goes to users *tom*, *patty*, and *trg*. The section "Creating Aliases," later in this chapter, tells how to set up aliases.

Specifying a Subject on the Command Line

The **–s** option allows you to include the subject of your message on the command line. To use this option, type **mailx –s**, then type the subject. If the subject contains more than one word, you must enclose it in quotation marks.

```
$ mailx –s "meeting time" wildcat
```

When you use this option, **mailx** will not prompt you for a subject when you type your message.

10.5.3 Working in Input Mode

When **mailx** is in input mode, you can edit or cancel messages, incorporate existing text into messages, change the header information, and perform other tasks that take advantage of **mailx** capabilities. The commands that perform these tasks are called tilde escapes. They consist of a tilde (~) followed by a letter or other character. You must type these commands at the beginning of a line and then press Return.

If you want **mailx** to interpret a tilde as text rather than as a command, type two tildes (~~). This is necessary only when the tilde is the first character on a line.

The following sections describe most of the tilde escape commands. To see a complete list, type the following command at the beginning of a line:

~? Return

Canceling Mail

If you decide not to send the message you are typing, you can use either the **~q** or the **~x** command to cancel it. Both of these commands cancel the message and return you to the shell. However, when you use **~q**, your text is saved in a file called *dead.letter* in your home directory. Only the most recently canceled message is stored in this file. If you cancel another message, that message will override the current contents of the *dead.letter* file.

The **~x** command does not save anything. It simply cancels the message and exits **mailx**.

Editing Mail

When **mailx** is in input mode, you can invoke a text editor by entering the ~e or ~v escape at the beginning of a line. The ~e command invokes the editor specified by the **EDITOR** environment variable (typically **ed**); the ~v command invokes the editor specified by the **VISUAL** environment variable (typically **vi**). In the following example, we used ~v to start up the **vi** editor.

```
$ mailx wildcat Return
Subject: Meeting agenda Return
~v Return
```

We then used **vi** commands to type the message. When we exited from **vi**, **mailx** returned to input mode and prompted us to continue the message.

```
Sun bathing   10:00
~
~
"tmp/Re05069" 3 lines 60 characters
(continue)
```

Previewing a Message

The ~p escape allows you to preview the message you're creating. It will display the entire message up to the point where you typed ~p. On the following screen, we used ~p to look at the message we created previously.

```
~p Return
Message contains:
To: wildcat
Subject: Meeting agenda

Stretching          9:00
Grooming            9:30
Sun bathing        10:00
(continue)
```

Incorporating Existing Text

mailx provides several tilde escapes you can use to incorporate material from another source into the message you are creating:

~r	Reads a file into your message
~f	Reads mail you have received into a message (useful for forwarding mail to another user)
~m	Reads mail into a message and indents the mail one tab stop (useful for forwarding mail to another user)
~d	Reads the contents of the *dead.letter* file into your message
~i	Incorporates the value of an environment variable into your message
~<!	Incorporates output from a shell command into your message

Reading a File into a Message. The **~r** escape reads the contents of an existing file into your message. To use this command, type **~r** followed by the filename, as shown on the following screen.

```
$ mailx daves (Return)
Subject: Schedule (Return)
Here is the schedule for next week: (Return)
~r letters/schedule  (Return)
"letters/schedule"   10/725
```

The system then prints the filename and the number of lines and characters it contains (in this example, 10 lines and 725 characters). The **mailx** program is still in input mode and you can continue typing the rest of the message. (If you want to look at the contents of the file, use the **~p** escape to preview the message.)

Forwarding or Incorporating Mail. The **~f** escape incorporates a message from your mailbox into the message you are creating. To use this escape, **mailx** must first be in command mode (where you read your mail). At the **mailx** command mode prompt (?), type **mail** or **m** followed by the logins of the users you want to receive your message. You can then specify a subject and begin typing your message.

When you want to incorporate a message from your mailbox, type ~f followed by either the message number or the login of the message sender. If you don't specify a message, the current message is incorporated. On the following screen, we typed **~f 2** to incorporate message 2.

```
$ mailx Return
mailx version 3.1  10/9/88  Type ? for help.
"usr/mail/daves": 2 messages 1 new
     1 abc        Tue May 1  08:09  8/155  Meeting Notice
 >   2 jim        Mon Apr 30 16:57  4/127  Schedule
? m jones Return
Subject: Next week's schedule Return
Here is a copy of Jim's schedule:  Return

~f 2 Return
Interpolating: 2
(continue)
```

mailx tells you it is incorporating message 2 and then asks you to continue. When you send the message, **mailx** returns to command mode, as shown by the **mailx** prompt (?) that appears on the screen. You can now continue in command mode or exit **mailx**.

The **~m** escape works in the same manner as **~f**. However, it indents the message being incorporated by one tab stop.

Incorporating an Environment Variable. The **~i** escape allows you to incorporate the value of an environment variable into a message. To use this command, type **~i** followed by the name of the variable. For example, the following command will print the name of your home directory (stored in the **HOME** variable) in your message.

```
~i HOME  Return
/user1/wildcat
```

Incorporating Command Output. The **~<!** escape allows you to include command output in a message. To use this escape, type **~<!** followed by the name of the command. The following command prints the current date in a message.

```
~<! date  Return
"date"  1/29
```

The system will display the name of the command that was executed, followed by the number of lines and characters in the command output (1 line and 29 characters in our example). However, the actual command output appears in your message. If you want to look at the output, use the **~p** escape.

Changing the Message Header

A **mailx** message header has four components:

- Subject
- Recipients
- Copy-to list
- Blind-copy list (a list of recipients that does not appear on the copies sent to other recipients)

When you are typing a message, you can use the following tilde escapes to change the header information. The commands must begin in the first position on a line.

Change All Header Information. The **~h** escape lists the header fields and their current values one at a time. You can add to or change each value. To accept the current value, simply press Return. The following screen shows the fields that appear when you use the **~h** command.

```
~h  Return
To: wildcat puma  Return
Subject:  Meeting agenda  Return
Cc:  panther cougar  Return
Bcc:  Return
(continue)
```

On this screen, we added another login name to the recipient line and to the `Cc:` line, and accepted the remaining values.

Change the Subject. The **~s** escape allows you to change the subject line while you are typing a message. To use this command, type **~s** followed by the new subject.

~s Revised meeting agenda Return

The original subject line does not appear on the screen.

Change the List of Recipients. The **~t** escape allows you to add additional login names or aliases to the list of recipients. To use this command, type **~t** followed by the new login names or aliases.

~t bobcat Return

The original recipients do not appear on the screen.

Change the Copy-To List. The **~c** escape allows you to add login names or aliases to the copy-to list for the message. To use this command, type **~c** followed by the login names or aliases.

~c tiger Return

The original names on the copy-to list do not appear on the screen.

Change the Blind-Copy List. The **~b** escape allows you to add login names or aliases to the blind-copy list for the message. To use this command, type **~b** followed by the login names or aliases.

~b gazelle impala Return

The original names on the blind-copy list do not appear on the screen.

Adding Your Signature

The **mailx** program includes two environment variables, **Sign** and **sign**, that allow you to establish signatures you can include in messages. You can assign values to these variables in your *.mailrc* file, which is described later in this chapter. The **~a** command inserts the value of **sign** into your message; the **~A** command prints the value of **Sign**. For example, if the value of the **Sign** variable is "Supreme Commander," that value will be added to your message when you type **~A**.

```
$ mailx -s orders all Return
Be ready to move out at 0400 hours. Return
~A Return
Supreme Commander
```

Because your login automatically appears in the header of each message you send, you don't need to use a signature to identify yourself.

Returning to the Shell

The **~!** escape allows you to leave **mailx**, execute a shell command, and then return to the same place in **mailx**. To use this command, type **~!** followed by the name of the command, as shown on the following screen.

```
during the blackout.  Return
~! ls  Return
memos
letters
reports
!
```

When the command finishes executing, an exclamation point (!) appears on the screen. You are now back in **mailx** and can continue your message.

Executing Command Mode Commands

The **~:** escape takes **mailx** out of input mode, allowing you to execute a **mailx** command mode command. These commands are described later in this chapter.

Keeping Copies of Your Messages

The **mailx** program offers you several ways to keep copies of outgoing messages. The **~w** escape and the **-F** command line option both save the text of a message, however only the **-F** option saves the header information.

To save the text of your message, type **~w** followed by a filename. For example, the command on the following screen saves the text in a file named *savemail*.

```
When you want to save a copy of Return
your text, use the tilde w. Return
~w savemail  Return
"savemail" 2/58
```

If you want to save the header information as well as the text, use the **-F** option on the command line. This option places a copy of the message in a file named after the first recipient of the message. If your current directory already has a file by that name, the text of the message is appended to it. For example, the following command will save the message in a file named *sam*.

```
$ mailx -F sam puma lynx Return
```

If you specify an alias as the recipient, **mailx** converts the alias into the appropriate logins and then uses the first login as the filename. For example, if the *mfg* alias includes the logins *sarah*, *tommy*, and *kol*, the following command will save the message in a file named *sarah*.

```
$ mailx -F mfg Return
```

10.5.4 Managing Mail: Command Mode

If there are messages in your mailbox, the message `you have mail` appears on your screen when you log in or when you return to the shell from another procedure. To read your mail, type **mailx** and press Return. This command places **mailx** in command mode and prints a list of your messages. The following screen shows a sample list.

```
$ mailx  Return
mailx version 3.1  10/19/88    Type ? for help.
"/usr/mail/puma":   3 messages  3 new
> N 1 rbt          Thur Apr 30 14:20   8/190  Review
  N 2 admin        Thur Apr 30 15:56   5/84   New printer
  N 3 jim          Fri  May  1 08:39  64/1574 Recap
?
```

The first line identifies the version of **mailx** used on your system, prints the date, and reminds you to type a question mark if you need help. The second line shows the pathname to your mailbox file, the number of messages it contains, and the status of each message.

The remainder of the display contains header information about the messages. The messages are numbered; the last message received is at the bottom of the list. A status indicator appears to the left of the numbers:

N	The message is new.
U	The message is old (from a previous **mailx** session), but has not been read.
blank	The message has been read, but not saved.
*	The message has been saved.

A greater-than sign (>) points to the current message.

The remaining fields in each header line show the login of the message originator, the day, date and time the message was delivered, the number of lines and characters it contains, and its subject. The subject field might be blank.

The list of messages is followed by the **mailx** command mode prompt (?). You will enter your commands at this prompt.

To help you become familiar with reading and managing your mail, use the procedures described earlier in this chapter to send yourself a group of messages. You can then practice reading, deleting, and saving them.

Working in Command Mode

mailx has over fifty commands that help you manage your incoming mail. Many of these commands have abbreviations that you can use in place of the full command name. For example, you can type either **save** or **s** when you want to save a mail message. There are also duplicate commands for several functions. For example, both the **t** and **p** commands print messages on your screen.

Many **mailx** commands can take a list of messages as an argument. The command then operates on each message in the list. If a command expects a message list and you do not provide one, the command is performed on the current message.

In a message list, you can specify messages by number, author, subject, or type. The list can include any of the following formats:

n	A single message number, such as 4
.	The current message
^	The first undeleted message
$	The last message
*	All messages
n-m	An inclusive range of message numbers, such as 4-10
user	All messages from *user*
/string	All messages with *string* in the subject line (case is ignored)
:*c*	All messages of type *c* where *c* is one of the following:

d Deleted messages

n New messages

- **o** Old messages
- **r** Read messages
- **u** Unread messages

The context of the command determines whether this type of specification makes sense.

The examples in the following sections show how to use these formats.

If you need help while **mailx** is in command mode, type either a **?** or **help** after the **mailx** prompt (?) and press Return. A description of several **mailx** commands and their functions then appears on your screen. You can also type **list** at the **mailx** prompt to display a list of all **mailx** commands. This list does not include explanations of the commands.

Reading Mail

When you issue the **mailx** command, **mailx** lists header information for the messages in your mailbox. There are several commands that allow you to read these messages.

To read the current message (indicated by the greater-than (>) sign), type **n** and press Return, or just press Return.

? **n** Return

? Return

When you reach the end of the message, the **mailx** prompt (?) reappears. To read the next message, press Return again.

To read a different message, enter either **p** (for **print**) or **t** (for **type**) followed by the number assigned to the message. (The **t** and **p** commands are identical.) You can also omit the **t** or **p** and just type the message number. All of the following commands will print message number 2.

? **p 2** Return

? **t 2** Return

? **2** Return

If you want to print a series of messages, specify the appropriate range, such as **p 3-5**.

You can also use the login name of the sender to specify a message. To do this, follow the **t** or **p** command with the appropriate login name. For example, the following commands print all messages from user *daves*.

? **p daves** Return

? **t daves** Return

Exiting mailx

There are two commands you can use to leave **mailx**: **x** and **q**. The **x** command leaves your mailbox almost unchanged. If you use the **x** command to leave **mailx**, the messages you read or deleted remain in your mailbox. However, if you saved messages in other files, that action has already occurred and cannot be undone by the **x** command.

If you use the **q** command, **mailx** prints messages summarizing what you did with your mail. The messages look like this:

```
? q Return
Saved 1 message in /user1/joanie/mbox
Held 1 message in /usr/mail/joanie
$
```

The messages on the previous screen mean that user *joanie* had at least two messages, that she read one message, and either left the other message unread or issued a command to hold it in her mailbox, */usr/mail/joanie*. If there were more than two messages, *joanie* deleted them or saved them in other files. **mailx** does not issue a message about those situations.

By default, if you read a message and don't delete or save it, when you use **q** to leave **mailx**, the message is stored in a file called *mbox* in your home directory. As you read additional messages, they are added to the beginning of this file. You can use **mailx** commands to read, delete, or transfer the messages in your *mbox* file. You can also use **mailx** commands to save messages in files other than *mbox*.

The *.mailrc* file, described later in this chapter, allows you set up **mailx** to retain messages you have read in your mailbox or to create a directory to save mail messages.

Displaying Header Information

When **mailx** enters command mode, it prints a banner line listing the number of messages waiting in your mailbox. The header lines for up to 20 messages will also display. (If you dialed into the computer system, the header lines for up to 10 messages would appear.)

If you read a message and then want to look at the header lines again, type the following command and the header lines will reappear.

? **h** Return

If your mailbox contains more than 20 messages, you can use the **z** command to display the next screen of header lines. The following command shows the next screen.

? **z** Return

Typing **z–** shows the previous screen.

? **z–** Return

If you want to see the header information for a specific group of messages, use the **f** (or **from**) command. For example, the following command shows header information for messages 5 through 7.

? **f 5-7** Return

You can also use the login name of the message sender to specify the header information to print. The following command will display the header lines for all messages from user *daves*.

? **f daves** Return

If you are starting up **mailx** and want to look only at the header information, use the **mailx –H** option on the command line. This option shows only the header information. It doesn't allow you to read messages.

$ **mailx –H** Return

Reading the First Lines of a Message

The **t** (or **top**) command prints the first five lines of the messages that you specify. For example, to look at the first five lines of each of your messages, you would type this command:

```
? t * Return
```

If you want a different number of lines to appear when you type the **top** command, set the **toplines** variable in your *.mailrc* file, as described later in this chapter.

Reading Other Mail Files

When you issue the **mailx** command to read your mail, you are looking at the contents of your mailbox file. You can also use **mailx** to read mail messages stored in other files such as your *mbox* file, which stores messages that are read but not explicitly saved or deleted.

The **–f** command line option allows you to read messages from a file other than your mailbox. To read the messages stored in your *mbox* file, type this command:

```
$ mailx -f Return
```

If you are using a different file to store messages, follow the **–f** option with the name of the file. For example, if you are storing messages in the *message* file in your home directory, you would type this command:

```
$ mailx -f message Return
```

When **mailx** is in command mode, you can switch to another mail file, such as *mbox*, and then read the messages in that file. (You can even switch to a non-mail file, but if you try to use **mailx** commands, the message `No applicable messages` appears.) To switch to another file, type **fi** or **fold** (these commands are identical) at the **mailx** prompt (`?`), then type the name of the file. For example, if you are reading messages in your mailbox and want to switch to your *mbox* file, you would type the command shown on the following screen.

```
? fi mbox Return

Held 3 messages in /usr/mail/puma
"/user1/puma/mbox": 15 messages
```

The system prints a summary of the messages in the original file (mailbox */usr/mail/puma* in this example), followed by the header information for the new file (in this example, */user1/puma/mbox*).

You can use the following special characters to represent the filename in a **fi** command:

%	Represents the current mailbox
%login	Represents the mailbox belonging to the specified *login* (you must have permission to read the file)
#	Represents the previous file you were reading
&	Represents the current *mbox* file

For example, if you are reading the contents of your mailbox, you can type the following command to switch to your *mbox* file:

? **fi &** Return

To switch from your *mbox* file back to your mailbox, you could type either of these commands:

? **fi %** Return

? **fi #** Return

Deleting Mail

The **d** (or **delete**) command deletes messages. The messages are not actually removed until you leave the mail file you are processing. Before you switch to another mail file or exit **mailx**, you can use the **u** (or **undelete**) command to restore the mail you deleted.

To delete the current message, type **d** and press Return. If you want to delete a different message, follow the **d** command with the number of the message or the login of the message sender. The first command shown below deletes message 6; the second command deletes messages 8 through 12.

? **d 6** Return

? **d 8-12** Return

If you want to delete all of the messages you have read, type this command:

? **d :r** Return

The next command deletes all messages in your mailbox:

```
? d * Return
```

If you delete a message by mistake, you can use the **u** command to restore the message. For example, to restore message 10, type this command:

```
? u 10 Return
```

You can combine the delete and print commands into a single command: **dp**. This command deletes the current message and prints the next message.

```
? dp Return
```

Saving Mail

The **s** and **S** commands save messages in files other than your *mbox*. If the file you specify already exists, the message is appended to it. If the file does not exist, it is created.

The **S** (or **Save**) command saves the message you specify in a file in the current directory named for the sender of the message. If you specify a list of messages, the messages are saved in a file named for the sender of the first message in the list. If you don't specify a message, **mailx** saves the current message. For example, your mailbox might contain these messages:

```
mailx version 2.14  10/19/86    Type ? for help
"/usr/mail/puma":   3 messages  3 new
> N 1 rbt          Thur Apr 30 14:20   8/190   Review
  N 2 admin        Thur Apr 30 15:56   5/84    New printer
  N 3 jim          Fri  May  1 08:39  64/1574  Recap
?
```

The following command saves message 3 in a file named *jim*, the name of the message originator.

```
? S 3 Return
```

The **s** (or **save**) command allows you to specify a filename for the message you are saving. To use this command, type **s** followed by the message to be saved, then type a name for the file. For example, the following command saves message 2 in a file named *printer*.

```
? s 2 printer Return
```

If you don't specify a message, the current message is saved. The following command will save the current message (message 1 on the previous screen) in a file named *rbt.review*.

? **s rbt.review** Return

When you omit the message portion of the command, be careful about using a login name as the name of the file. If there are several messages from that sender, **mailx** might not be able to interpret your command. In such cases, you will receive an error message.

Copying Messages

When you use the **save** command, **mailx** deletes the message you saved from your mailbox. If you want to retain the saved message in your mailbox, use the **co** (or **copy**) command. This command copies a message to a file and retains the original message in your mailbox.

To use this command, type **co** followed by the message and a name for the file that will store the message. The following command places a copy of message 12 in a file named *parks*.

? **co 12 parks** Return

If you don't specify a filename, the copy will be saved in your *mbox* file. For example, the following command saves messages from *shiela* in your *mbox* file:

? **co shiela** Return

The **C** (or **Copy**) command copies the message you specify to a file named for the sender of the message. The following command saves message 2 in a file named for the message originator.

? **Copy 2** Return

Holding Messages in Your Mailbox

The **ho** (or **hold**) command saves the messages you specify in your mailbox instead of moving them to *mbox*. The **preserve** command also does this. For example, to retain message 4 in your mailbox, you could type either of these commands:

? **ho 4** Return

? **preserve 4** Return

Replying to Mail

The **R** and **r** commands can be used to reply to a mail message. The uppercase **R** (or **Reply**) sends your response only to the originator of the message; the lowercase **r** (or **reply**) sends your response to the originator of the message and to all other recipients of that message.

When you issue these commands, **mailx** uses the original subject line as the subject of your reply. On the following screen, we used the **R** command to reply to message 2.

```
$ mailx Return
mailx version 2.14 10/19/83  Type ? for help.
"usr/mail/daves":  3 messages 2 new 3 unread
    1 jaf          Wed May 9 07:55   7/137   test25
>   2 todd         Wed May 9 08:59   9/377   requirements
  N 3 has          Wed May 9 11:08  29/1214  bailey

? R 2  Return
To: todd
Subject: Re: requirements
```

If the `requirements` message went to some additional people and you would like them to receive your response, use the lowercase **r** to create your reply. The header information will look like this:

```
? r 2  Return
To: todd eg has jcb bdr
Subject: Re: requirements
```

Answering Mail and Saving Your Reply

The **f** (or **followup**) and **F** (or **Followup**) commands are similar to the **reply** command. The **f** command operates in the same manner as the **r** command, however it saves a copy of your response in a file named after the sender of the original message.

The **F** command allows you to reply to the first message in a list of messages. It then sends a copy of your reply to the sender of each message in the list. For example, the command shown on the following screen lets you reply to message 1 and sends the reply to the senders of messages 1 through 3.

```
? F 1-3 Return
To: jaf todd has
Subject: Re: test25
```

A copy of your reply is saved in a file named after the sender of the first message. The reply shown on the previous screen would be saved in a file named *jaf*.

Editing Messages

mailx allows you to edit any of the messages in your mailbox. The **e** (or **edit**) command invokes the **ed** line editor on the messages you specify. The **v** (or **visual**) command invokes the **vi** editor. For example, to use **vi** to edit message 3, you would type this command:

? **v 3** Return

Piping a Message Through a Command

The **pipe** (or **|**) command pipes a message through the command you specify. For example, you can can use this command to send a message through a text processing program or to send a message to your printer. Both of the following commands send message 3 to the printer.

? **pipe 3 lp** Return

? **| 3 lp** Return

Returning to the Shell

There are two commands that allow you to return to the shell: **!** and **sh**. The exclamation point (!) allows you to return to the shell and execute a single command. You then return to **mailx**. To use this command, type an exclamation point at the **mailx** command prompt (?), then type the command you want to execute.

```
? !date Return
Wed Jul 27 14:30:08 PDT 1988
!
?
```

When the command finishes executing, an exclamation point appears on your screen, followed by the **mailx** prompt (?).

The **sh** (or **shell**) command starts up a new shell, allowing you to enter as many commands as you want. When you finish working in the shell, type **exit** or Ctrl-D and you will return to **mailx**.

10.5.5 Mailing Files

If you want to mail a file to another person, use the name of the file as input for the **mailx** command. The following command mails the *memo* file to *marie*.

```
$ mailx marie < memo Return
```

You can also use the **~r** command, described earlier in this chapter, to read a file into the message you are typing. On the following screen, we used the **~r** command to read the *memo* file into a message for user *marie*.

```
$ mailx marie  Return
Subject: Memo  Return
Here's a copy of my memo:  Return

~r memo  Return
```

10.5.6 Setting Up Your mailx Environment

The **mailx** program includes many environment variables that determine how the program operates. Your system probably contains a **mailx** start-up file named */usr/lib/mailx/mailx.rc*. The system administrator uses this file to set **mailx** variables for the entire system. When you invoke **mailx**, the system uses the values in this file to establish your **mailx** environment.

If you want to modify your environment, you can change **mailx** variables for the current **mailx** session only, or you can change the variables permanently.

Displaying Variables

The **mailx** program includes two types of environment variables. One type of variable is either *enabled* (on) or *disabled* (off). For example, the **askcc** variable, which causes the `Cc:` prompt to appear when you create a message, is either enabled or disabled. The second type of variable takes a specific value. For example, to set the **crt** variable, you need to specify the number of lines that can be displayed on your screen.

You can use the **mailx set** command to display variables that have been enabled or assigned values. While **mailx** is in command mode, type **set** at the **mailx** prompt (`?`). The variables that appear will look something like this:

```
? set  Return
DEAD="/user1/puma/dead.letter"
MAILRC="/user1/puma/.mailrc"
MBOX="/user1/puma/mbox"
askcc
asksub
dot
folder="mymail"
header
metoo
save
```

Modifying Variables

You can use the **set** and **unset** commands to change the **mailx** variables. For most variables, the default settings should be fine. However, this chapter lists several variables you might want to change.

If a variable is off and you want to turn it on, type **set** followed by the name of the variable. For example, the following command enables (or turns on) the **dot** variable, which allows you to end a message by typing a period in the first position on a line.

set dot

To disable (or turn off) a variable, type **unset** followed by the name of the variable. The following command turns off the **asksub** variable, which causes the `Subject` prompt to appear when you create a message.

unset asksub

If the variable requires a value, type a command in this form:

set *variable=value*

The following command changes the **mailx** prompt from a question mark (`?`) to a colon (`:`).

set prompt=:

If a value contains spaces, enclose it in quotation marks.

You can set or unset several variables on the same line. For example, the following command sets the **dot** and **asksub** variables.

set dot asksub

Modifying Variables for the Current Login Session

To set or unset a variable for the current **mailx** session only, type the **set** or **unset** command at the **mailx** prompt (`?`). For example, the following command sets the **askcc** variable for the current **mailx** session.

`?` **set askcc** Return

Modifying Variables Permanently

If you want to change **mailx** variables permanently, use an editor such as **vi** to create a file named *.mailrc* in your home directory. You can also copy an existing *.mailrc* file from another user on your system.

Figure 10-1 contains a sample *.mailrc* file. This file is typically used to set or unset **mailx** environment variables and to define **mailx** aliases.

```
set append askcc autoprint dot
set metoo quiet header hold keepsave
unset asksub save
set outfolder
set folder='mymail'
set record='mymsgs'
set crt=22
set sign='Roberts'
set Sign='Jack Roberts'
set toplines=10
alias amandap ap
alias robsgrp rob fred bob alice pat mark
alias accounts   robsgrp donna
```

Figure 10-1. Sample *.mailrc* file.

If you create a *.mailrc* file, the system reads it whenever you start up the **mailx** program and adjusts your **mailx** environment according to the values it contains. If the system-wide *mailx.rc* file and your *.mailrc* file contain different settings for the same variable, the setting in your *.mailrc* file takes precedence.

If your *.mailrc* file redefines the variables assigned in the system-wide file, you can use the following command line option when you invoke **mailx**:

```
$ mailx -n Return
```

This option tells **mailx** not to read the system-wide *mailx.rc* file. Instead, it goes directly to your *.mailrc* file, resulting in faster initialization.

mailx Environment Variables

This section describes several **mailx** environment variables that you might want to include in your *.mailrc* file.

Setting Up Folders. The **folder** environment variable allows you to set up a directory to store the messages you receive. The files in this directory are called *folders*.

To set up a directory for storing mail, first use the **mkdir** command to create the directory. Then assign the name of the directory as the value of the **folder** variable. For example, if you named your directory *mymail*, you would add this line to your *.mailrc* file:

```
set folder=mymail
```

The system assumes the *mymail* directory is directly below your home directory. If your mail storage directory is in another location, be sure to include the appropriate pathname when you set the **folder** variable.

To save a message in your mail storage directory, type the **s** command followed by the message. Then type a plus sign (+) and the name of the file that will store the message. For example, to save message 2 in a file named *jim*, you would type this command:

```
? s 2 +jim Return
```

The plus sign (+) tells **mailx** to place the message in the directory indicated by the **folder** variable. (In our example, the message is stored in the file *myfile/jim*.) If the file already exists, the message will be appended to it.

Saving Outgoing Mail. If you set the **record** variable, **mailx** will save copies of all messages you send. To set this variable, specify the file that is to store the messages.

```
set record=mymsgs
```

The **mailx** program will now save both the header information and the text of your messages in the *mymsgs* file. You can use the **mailx** commands to read or delete the messages in this file.

If you want to place your outgoing mail file in your mail storage directory (indicated by the **folder** variable), set the **outfolder** variable in your *.mailrc* file.

```
set outfolder
```

When this variable is set, **mailx** will save a copy of all outgoing mail in *mymail/mymsgs*.

Prompting for a Subject. The **asksub** variable determines whether the `Subject:` prompt appears when you are creating a message. This variable is enabled by default. If you want to disable it, add the following line to your *.mailrc* file or type it at the **mailx** command prompt.

```
unset asksub
```

Using a Dot to Exit mailx. When you are creating a message, you can type ~. to end the message and cause **mailx** to leave input mode. If you want to use a dot (.) for this purpose, set the **dot** variable, as shown below, either in your *.mailrc* file or on the command line.

```
set dot
```

Saving Canceled Messages. The **save** variable saves canceled or undeliverable messages in the *dead.letter* file in your home directory. This variable is enabled by default. If you don't want to save canceled or undeliverable messages, unset this variable on the command line or in your *.mailrc* file.

```
unset save
```

Naming the Dead Letter File. **mailx** saves canceled or undeliverable mail in a dead letter file. By default, this file is named *dead.letter*. If you want to save this mail in a different file, set the **DEAD** variable. For example, if you add the following line to your *.mailrc* file, canceled or undeliverable mail will be saved in the *cancel.mail* file in your home directory.

```
set DEAD=$HOME/cancel.mail
```

Prompting for Carbon Copies. When you finish typing a message, **mailx** can display a `Cc:` prompt where you can type the names of users you want to receive a copy of the message. The **askcc** variable (which is disabled by default) causes this prompt to appear. If you want the `Cc:` prompt to appear when you finish typing a message, set the **askcc** variable on the command line or in your *.mailrc* file.

```
set askcc
```

Replacing the Tilde Escape. The **mailx** input mode commands begin with a tilde (~). You can use the **escape** variable to transfer this function to another character. For example, the following line changes the escape character from a tilde to a colon (:):

```
set escape=:
```

You would then begin input mode commands with a colon, rather than a tilde.

Defining Your Signature. The **sign** and **Sign** variables allow you to define two signatures. The following lines assign the value `topcat` to the **sign** variable and the value `top dog` to the **Sign** variable. (Note that when a signature contains spaces, it must be enclosed in quotation marks.)

```
set sign=topcat

set Sign='top dog'
```

You can use these variables when **mailx** is in input mode. The command **~a** inserts the value of the **sign** variable into your message. The command **~A** inserts the value of the **Sign** variable.

Suppressing the Opening Line. When you invoke **mailx** to read messages, the first line on the screen contains the **mailx** version number. If you don't want to see this information, you can set the **quiet** variable in your *.mailrc* file.

```
set quiet
```

Disabling the Header Display. When you invoke **mailx** to read mail, header information about the messages in your mailbox will appear on the screen. The **header** variable causes this information to print. If you don't want to see the header information, unset the **header** variable, as shown below.

```
unset header
```

Specifying the Number of Header Lines to Print. The **screen** variable allows you to specify the number of header lines you want to print on the screen. For example, if you want to see 15 lines, add this line to your *.mailrc* file:

```
set screen=15
```

Changing the mailx Prompt. By default, **mailx** uses a question mark (?) as its prompt. The **prompt** variable allows you to change the prompt to another character. For example, the following line changes the prompt to a percent character (%) followed by a space. Quotation marks are necessary to include the space.

```
set prompt='% '
```

Specifying a Pager Program. When you read a long message, **mailx** invokes a paging program such as **pg**, which is the default. If you want to invoke a different program, or invoke **pg** with one of its options, set the **PAGER** variable. The following line invokes **pg** with the **-s** option, which prints messages and prompts in standout mode (usually reverse video).

```
set PAGER='pg -s'
```

Specifying the Number of Lines Per Page. The **crt** variable specifies the minimum number of lines that can be displayed before **mailx** invokes the pager program. For example, if your terminal can display 22 lines, you would set the **crt** variable as shown below.

```
set crt=22
```

Printing a Specific Number of Lines from a Message. The **top** command prints the first five lines of the message you specify. If you want **top** to print a different number of lines, set the **toplines** variable. If you add the following line to your *.mailrc* file, **top** will print 15 lines.

```
set toplines=15
```

Combining the delete and print Commands. The **autoprint** variable combines the **delete** and **print** commands. If you enable this variable, when you delete a message, the next message will appear automatically. To use the **autoprint** option, add this line to your *.mailrc* file:

```
set autoprint
```

Keeping Messages in Your Mailbox. When you leave **mailx**, the messages you read are transferred to your *mbox* file and deleted from your system mailbox. If you want to keep all messages in your mailbox unless you explicitly save or delete them, set the **hold** variable:

```
set hold
```

The messages you've explicitly saved are removed from your system mailbox when you exit **mailx**. If you want to keep a copy of saved messages in your system mailbox, enable the **keepsave** variable:

```
set keepsave
```

If you enable both the **hold** and the **keepsave** variables, all messages will remain in your system mailbox until you delete them.

Specifying a Default for the Pipe Command. The **pipe** command allows you to process messages through another command. For example, the command **pipe 3 lp** sends message 3 to your printer. If you always use **pipe** with the same command, you can set that command as the value of the **cmd** variable. For example, the following line assigns **lp** as the value of this variable.

```
set cmd=lp
```

The **pipe** command now automatically processes messages through the **lp** command. You no longer need to include **lp** when you issue a **pipe** command. For example, the command **pipe 3** will send message 3 to the printer.

Inserting Form Feeds. If you use the **pipe** command to send messages to your printer, you might find the **page** variable useful. This variable inserts a form feed between the messages you send to the printer, causing each message to start printing on a new page. To enable the **page** variable, add this line to your *.mailrc* file:

```
set page
```

Appending Messages to Your *mbox* File. When **mailx** transfers a message to your *mbox* file, the message is added to the beginning of the file. The most recent message appears first; the oldest message appears last. If you want the messages in your *mbox* file to be in the order they were received (oldest message first, newest message last), set the **append** variable:

```
set append
```

Now, messages sent to *mbox* will be added to the end of the file.

Receiving Messages You Send. If you send a message to a group alias and you belong to the group, **mailx** will remove your name from the list of recipients before it sends the message. Everyone belonging to the alias will receive

the message except you. If you want to receive a copy of messages you send to aliases to which you belong, set the **metoo** variable:

```
set metoo
```

Creating Aliases

An alias is a word that represents one or more login names. If you frequently send mail to a group of users, such as the members of a department, you can create an alias that includes the logins of those users. When you send mail to an alias, the message goes to all users belonging to that alias.

To create an alias, type **alias** followed by the name that will represent the alias. Then type the login names of the persons to be included in the alias. For example, to create an alias named *accts* for users *mjf*, *arv*, and *janet*, you would add this line to your *.mailrc* file:

```
alias accts mjf arv janet
```

Now you can send mail to these users by typing the following command:

```
$ mailx accts Return
```

You can also create an alias for a single user. You might want to do this if a user has a long login name. For example, you could create the alias *tm* for user *thompson*.

An alias can include another alias. In the *.mailrc* file shown in Figure 10-1, the alias *accounts* includes the alias *robertsgroup* and the login name *donna*. Therefore, the *accounts* alias expands to seven logins.

Adding Other Commands to a *.mailrc* File

You can add most **mailx** commands to your *.mailrc* file. As an example, if you add the following line to your *.mailrc* file, the `Status` line will not appear in the header information when you read your mail messages.

```
ig status
```

The following commands cannot be used in the *.mailrc* file: **!**, **C**, **e**, **f**, **F**, **h**, **m**, **p**, **r**, **R**, **sh**, and **v**.

Other Information

mailx(1)

10.6 Using UUCP

UUCP (short for *UNIX to UNIX copy*) is a collection of programs that allow you to communicate with users on other systems. Both the **mail** and **mailx** programs use UUCP to send messages to users at other sites. You can use UUCP to transfer files between machines and to execute commands on another machine.

When you issue a UUCP command, it is not executed immediately. Instead, the UUCP programs *queue*, or schedule, your requests for execution at a later time. If you are transferring a file, a copy of the file is *spooled* or placed in a temporary directory. After UUCP transmits a copy of the file, it deletes the file from the temporary directory.

The procedures at your site determine how often UUCP commands execute. Some sites execute UUCP commands immediately; other sites execute them at specific intervals, perhaps only at night. Some sites do not have the equipment necessary to call remote systems. These systems wait for the remote system to call them and then execute the queued commands.

10.6.1 UUCP Communications

For the UUCP system to work, the computers using it must be able to communicate. If the computers are at the same site, they can be hard-wired to each other or connected by a local area network such as Ethernet. If the computers are at different sites, the modem on one computer must be able to connect, or dial, the modem on the other computer over the telephone lines.

10.6.2 Communicating with Users on Other Systems

To communicate with a user on another system, you need to know the user's login name, the name of the user's system, and whether your system communicates with that system.

Finding the Name of Another Site

If you can, obtain from the recipient the name of the remote system and the recipient's login name. If the recipient does not know the system name, ask the recipient to issue the **uname –n** command on the remote system. The name of the system will then appear.

```
$ uname -n  Return
sys2
$
```

Determining if Your Site and a Remote Site Can Communicate

When you know the name of the remote system, use the **uuname** command to determine if your system can communicate with the remote system. This command lists the systems that can communicate directly with your site.

```
$ uuname Return
```

If the list is quite lengthy, you can use the **grep** command to search through the **uuname** output. If **grep** finds the specified system name, it prints the name on your screen.

```
$ uuname | grep sys2 Return
sys2
$
```

If the remote system does not communicate with your system, **uuname** returns only the shell prompt.

Other Information

uname(1C), **uucp**(1C) (includes **uuname**)

Sending Mail via UUCP

You can use either **mail** or **mailx** to send messages to a user on a system that communicates directly with your system. To specify the user, type the system name, an exclamation point, and then the user name. For example, to send a message to user *wildcat* on system *feline*, you would type this command:

```
$ mailx feline!wildcat Return
```

If you are using the C shell, precede the exclamation point with a backslash, as shown below. This is necessary because the exclamation point has a special meaning to the C shell program. The backslash negates this meaning.

```
$ mailx feline\!wildcat Return
```

If your site belongs to the UNIX network, you might also be able to communicate with sites you cannot reach directly. This is done by sending your message to a system you can reach. That system forwards your message, either to the system you are trying to reach, or to another system that will forward the message in turn. Your system administrator can help you determine the correct address for the message.

10.6.3 Transferring Files to Remote Systems

NOTE

The system administrator on the remote system must give your machine permission to execute specific commands on the remote system. Your system administrator can tell you which commands you can execute on the remote system.

The **uucp** and **uuto** commands transfer files to a remote system. These commands are useful when you need to send large files. **uucp** sends files to the directory of your choice on the remote system; **uuto** sends files to a public directory on the remote system.

The **uuto** command is an enhanced version of **uucp**. It is easier to use, but less private. Before you can issue the **uucp** command to transfer a file to another user's directory, the user must give you write permission on that directory. Because the **uuto** command automatically sends files to a public directory (called */usr/spool/uucppublic*) on the remote system, you don't need to obtain write permission for the destination directory. Also, the **uuto** command line is shorter and less complicated than the **uucp** command line.

Before using **uucp** or **uuto** to transfer a file, you should check the permissions on the file to determine if it can be transferred. You can transfer a file only if it meets these criteria:

- The file must have read (**r**) permission for others.
- The directory containing the file must have read (**r**) and execute (**x**) permission for others.

If these permission levels are not present, use the **chmod** command to set them. After you transfer a copy of the file, you can replace the previous permissions.

Transferring Files to a Specific Directory: uucp

The **uucp** command allows you to send one or more files to a directory on a remote system. It is also possible to use **uucp** to send a file from a remote system to your local system. However, for security reasons, you will probably not be allowed to do this. Ask someone on the remote system to send the file to you instead.

The process of transferring a file is called a *job*. When you issue the **uucp** command, your file is not sent immediately. Instead, your command is placed in a queue of waiting jobs and assigned a job number. When your system executes the commands in the queue, the file transfer takes place. When the file arrives at the remote system, the recipient receives a mail message regarding the transfer.

The **uucp** command has this format:

uucp *filename system!destination-file*

filename is the name of the file to be sent, *system* is the name of the remote system, and *destination-file* is the name the file will have on the remote system.

To transfer your file, **uucp** must know the full pathname of both the file to be transferred and the *destination-file*. If the file being transferred is in your current directory, you can specify it by its name alone (without a path). If the file is in another directory, you must specify its full pathname.

To specify the destination for the file, type the name of the remote system followed by an exclamation point. (If you are using the C shell, precede the exclamation point with a backslash (\!).) Then type the full pathname for the file or use one of the abbreviations described in the following sections.

Transferring a File to a Private Directory. When you specify a private directory as the destination for the **uucp** command, you can type the full pathname for the file, or you can use the *~login* abbreviation, which expands to the name of the recipient's home directory.

For example, user *kol* is working on system *mickey* and wants to send a file called *chap1* (in the directory */usr/kol/text*) to user *wsm* on system *minnie*. Because user *kol* is currently working in */usr/kol/text*, he can specify the relative pathname, *chap1*, as the source file and use a full pathname for the destination file.

```
$ uucp chap1 minnie!/usr/wsm/receive/chap1 Return
```

User *kol* can also specify the destination by using the *~login* abbreviation followed by a name for the new file. The following command sends the *chap1* file to the *minnie!/usr/wsm/receive* directory and names the new file *chap2*.

```
$ uucp chap1 minnie!~wsm/receive/chap2 Return
```

If you don't supply a name for the destination file, **uucp** will give the destination file the same name as the source file. For example, the following command names the new file *chap1*.

```
$ uucp chap1 minnie!~wsm/receive Return
```

Transferring a File to the Public Directory. The **uucp** command can also transfer a file to the public directory (*/usr/spool/uucppublic*) on a remote system. (You can also use **uuto** to do this.) Use a full pathname to specify the destination.

```
$ uucp chap1 minnie!/usr/spool/uucppublic/wsm/chap1 Return
```

uucp Options

The **uucp** command has several useful options. If you include the **–m** option on the command line, you will receive a mail message stating whether the transfer succeeded. The **–s** option, followed by a filename, asks the program to place a status report of the file transfer in the specified file. Be sure to include a filename after the **–s** option. If you don't, the message `uucp failed completely` will appear and the transfer will not take place. The **–j** option displays the job number assigned to the transfer.

The command on the following screen uses all of these options. It sends a file called *minutes* to user *gws* on a remote computer named *eagle*.

```
$ uucp -m -s status -j minutes eagle!/usr/gws/minutes
eagleN3f45
$
```

The **–s** option specifies that a status report be placed in the file *status*. The **–j** option causes the job number assigned to this job (`eagleN3f45`) to be displayed.

The **–m** option indicates that you will be sent mail regarding the transfer. If **uucp** does not notify you of a successful transfer soon after you send a file, do not assume that the transfer failed. Not all systems equipped with networking software have the hardware needed to call other systems. These systems wait until a remote system calls them before they execute the queued commands.

Other Information

uucp(1C)

Transferring Files to a Public Directory: uuto

The **uuto** command transfers files to the public directory on another system. The command has this format:

uuto *filename system*!*login*

filename is the name of the file to be sent, *system* is the recipient's system name, and *login* is the recipient's login name. If you are using the C shell, precede the exclamation point with a backslash (\!).

The process of sending a file is called a *job*. When you issue a **uuto** command, your job is placed in a queue and assigned a job number. When your system executes the commands in the queue, it transmits the file to the public directory on the remote system. When the file arrives, a **mail** message is sent to the recipient, who must use the **uupick** command (described later in this chapter) to retrieve the file.

For example, to send the file *money* to login *marie* on system *sys2*, you would enter this command:

```
$ uuto money sys2!marie  Return
```

If you add the **–m** option to the **uuto** command line, you will receive a mail message when the job reaches the recipient's system.

```
$ uuto –m money sys2!marie  Return
```

The message you receive will look something like this:

```
$ mail Return
From uucp Thur Apr3 09:45 EST 1986
file /sys1/kol/money, system sys1
copy succeeded
?
```

Other Information

uuto(1C)

10.6.4 Checking Job Status: uustat

The **uustat** command tracks **uucp** and **uuto** jobs and reports on their status. If you type **uustat** and press Return, a status report for your **uucp** or **uuto** jobs will appear. Following is an example of the output **uustat** produces.

```
$ uustat Return
1145 kol sys2 10/05-09:31 10/05-09:33 JOB IS QUEUED
1220 kol sys2 10/05-09:45 10/05-09:33 JOB IS QUEUED
$
```

A status message contains this information:

- The job number assigned to the job (`1145` and `1220`)
- The login name of the person requesting the job (`kol`)
- The name of the recipient's system (`sys2`)
- The date and time the job was queued (`10/05-09:31` and `10/05-09:45`)
- The date and time this **uustat** message was sent (`10/05-09:33`)
- The status of the job (the jobs are queued but have not been sent)

To display a status report for a specific **uuto** job, include the **-j** option, followed by the job number. The command on the following screen prints a report for job 1145.

```
$ uustat -j1145 Return
1145 kol sys2 10/05-09:31 10/05-09:37 COPY FINISHED,JOB DELETED
$
```

This status report shows that the job was sent and then deleted from the job queue. It is now in the public directory of the recipient's system.

You can also use **uustat** to kill a job. To do this, use the **-k** option followed by the job number. For example, to kill job 1150, you would type this command:

```
$ uustat -k1150 Return
```

Other Information

uustat(1C)

10.6.5 Receiving Files: uupick

When a user on another system uses **uuto** to send you a file, you will receive a mail message when the file reaches the public directory on your system. Continuing the example from the previous section, user *marie* would receive the following mail message when the file *money* arrives in her system's public directory:

```
$ mail Return
From uucp Wed May 14 09:22 EST 1986
/usr/spool/uucppublic/receive/marie/sys1//money from sys1!kol arrived
$
```

The message contains this information:

- The first line tells when the file arrived.
- The second line, up to the two slashes (//), gives the pathname to the location of the file in the public directory.
- The part of the line following the two slashes contains the name of the file and the login name of its sender.

You can use the **uupick** command to print, move, or delete files from the public directory. When you type **uupick**, it searches the public directory for files sent to you. If **uupick** finds files waiting for you, it reports the first filename and then prints a question mark (?) prompt, as shown on the following screen.

```
$ uupick Return
from system sys1: file money
?
```

The following paragraphs describe several commands you can enter at the **uupick** prompt (?) to read, move, or delete your mail messages. To see a summary of these commands, type an asterisk (*) after the **uupick** prompt.

After you tell **uupick** how to handle the current file, a message about your next file in the public directory will appear. If there are no more files for you, the **uupick** prompt (?) appears.

Printing Files

To print the contents of a file on your screen, type a **p** after the **uupick** prompt (?).

```
?
p Return
```

If you received a directory, the names of the files in the directory will appear.

Moving Files

To move a file from the public directory and place it in your home directory, type an **m** after the **uupick** prompt (?).

```
?
m Return
```

To put the file in another directory, follow the **m** command with the name of the directory. The following command places the file in the *finances* directory.

```
?
m finances Return
```

To move all of your files from the public directory to your home directory, type an **a** after the **uupick** prompt (?).

```
?
a Return
```

If you want to place the files in another directory, type the name of the directory after the **a** command. The following command places the files in the *messages* directory.

```
?
a messages Return
```

Deleting Files

If you don't want to save a file, you can delete it from the public directory by typing **d** after the **uupick** prompt (?).

```
?
d Return
```

Skipping a File

If you don't want to do anything to a file now, press Return after the **uupick** prompt (?). The file will remain in the public directory until the next time you use the **uupick** command.

```
?
Return
```

Exiting uupick

To exit from **uupick**, type a **q** after the **uupick** prompt (?).

```
?
q Return
$
```

Other Information

uuto(1C) (includes **uupick**)

10.6.6 Executing Commands on a Remote System: uux

The **uux** command (short for *UNIX-to-UNIX system command execution*) allows you to execute operating system commands on remote computers. It can gather files, execute commands, and send standard output to a file on a specified computer. For security reasons, you might not be able to execute certain commands on the remote machine. You will receive a **mail** message if you try to execute a restricted command. Frequently you will be able to run only the **mail** and **mailx** commands.

The **uux** command has this syntax:

uux [*options*] *command_string*

You must delay the interpretation of special shell characters (such as <, >, |, and ^) in the *command_string*, either by placing quotation marks around the entire *command_string* or by typing a backslash before the special character.

The *command_string* usually includes these components:

system!command

system is the name of the remote system where you will execute the command; *command* is the program to be executed. (If you are using the C shell, precede the exclamation point with a backslash (\!).) For example, if your computer is hard-wired to a host computer, you can use the following command to remotely print files that reside on your computer:

$ **pr minutes | uux -p host!lp** Return

This command will print the *minutes* file on the printer attached to the host computer. The **-p** option passes the standard input for the **uux** command (**pr minutes**) along to the **lp** command.

Other Information

uux(1C)

10.6.7 Connecting a Remote Terminal: ct

The **ct** command connects your computer to a remote terminal equipped with a modem and allows a user on that terminal to log in. When you issue the **ct** command, it dials the phone number of the modem. The modem must be able to answer the call automatically. When **ct** detects that the call has been answered, it allows a user to log in on the computer.

This command can be useful when issued from the remote terminal itself. If you are using a remote terminal that is far from your computer and want to avoid long distance charges, you can use **ct** to have the computer place a call to your terminal. Simply call the computer, log in, and issue the **ct** command. The computer will hang up the current line and call your remote terminal back.

The **ct** command has this format:

ct [*options*] *telno*

telno is the telephone number of the remote terminal.

For example, you might be logged in on a computer through a local terminal and want to connect a remote terminal to your computer. The modem on the remote terminal has the phone number 503-555-3497. To connect the terminal, you would issue this command:

$ **ct –h –w5 –s1200 9=503-5553497** Return

The equal sign (=) tells **ct** to wait for a secondary dial tone. The dash (–) represents a four-second delay following the area code.

The **–s** option specifies a dialer operating at a speed of 1200 baud. If a dialer is not available, the **–w5** option tells **ct** to wait for a dialer for five minutes before quitting. The **–h** option tells **ct** not to disconnect the local terminal (the terminal where you issued the command).

If you are working on a remote terminal and want to use **ct**, you must first call the computer and log in. Then issue the the **ct** command to tell the computer to call your terminal.

$ **ct –s1200 9=9323497** Return

The **ct** command will first determine if a dialer is available.

- If **ct** locates a dialer, it will display this message:

```
Allocated dialer at 1200 baud
```

- If no device is available, **ct** sends this message:

```
1 busy dialer at 1200 baud Wait for dialer?
```

  If you type **n** (no), the **ct** command exits. If you type **y** (yes), **ct** asks you how long it should wait:

```
Time, in minutes?
```

  Type the number of minutes and press Return. (You can also use the **–w** command line option to specify how long **ct** should wait for a dialer.)

The **ct** command next asks if you want to hang up the line connecting your remote terminal to the computer:

```
Confirm hangup?
```

If you type **y** (yes), you will be logged off the system and **ct** will call your remote terminal back. If you type **n** (no), the **ct** command exits without making the call and you remain logged in on the computer.

Other Information

ct(1C)

10.6.8 Connecting a Remote System: cu

The **cu** command connects your computer with a remote computer, allowing you to be logged in on both computers simultaneously. You can execute commands on either computer and transfer files between the computers without dropping the connection.

The method **cu** uses to make the connection depends on the information you specify on the command line. You must supply either the telephone number or the system name of the remote computer. The **cu** command takes either of these formats:

cu [*options*] *telno*

cu [*options*] *systemname*

telno is the telephone number of the remote computer. Equal signs (=) represent secondary dial tones and dashes (–) represent four-second delays. Dashes are useful when your number includes an area code. If you specify a telephone number, **cu** passes it to the automatic dial modem.

systemname is the name of the remote system. This name must be included in the UUCP *Systems* file. (To see the list of computers in the *Systems* file, run the **uuname** command.) When you specify a system name on the command line, **cu** first obtains the telephone number and baud rate from the *Systems* file and then searches for a dialer.

When **cu** makes the connection, the remote computer prompts you to log in and enter your password.

```
connected
login:
```

When you finish working on the remote system, first log off the remote system, then terminate the connection by typing a tilde followed by a period (~.). You will remain logged in on the local computer.

NOTE

The **cu** *command cannot detect or correct errors; data can be lost or corrupted during file transfers. After a transfer, you can check for loss of data by running either the* **sum** *or* **ls –l** *command on both the file that was sent and the file that was received. These commands report the total number of bytes in each file. If the totals match, your transfer probably was successful.*

cu Options

Following are some command line options you might find useful. The **–s**, **–n**, and **–l** options should not be used if you specify a *systemname*.

Specify baud rate

The **–s** *speed* option allows you to specify a baud rate. As an example, you might want to use a 1200 baud dialer to connect your computer to a remote computer called *eagle*. (The phone number for *eagle* is 847-7867.) To do this, you would type this command:

cu –s1200 9=8477867 Return

The **–s1200** option causes **cu** to use a 1200 baud dialer to call *eagle*. If you omit the **–s** option, **cu** uses a dialer at 300 baud, the default speed.

Specify a TTY line

The **–l** *line* option allows you to specify a TTY line. For example, if there is a direct link between your computer and the remote computer, you can use this option to indicate the line (port) associated with the direct link. In this case, no telephone number is necessary.

You can also use this option to specify a modem attached to a particular TTY line. For example, if your 2400 baud modem is connected to line `ttyGA/GABB`, you would type **cu –l ttyGA/GABB** *telno* (omit the */dev* prefix).

Prompt for a number

The **–n** option instructs **cu** to prompt you for a telephone number. This option provides additional security, as the telephone number does not appear in the output from the **ps** command. The **–n** option looks like this:

```
$ cu -n  Return
Number: 1234567  Return
```

Executing Commands

After you log in on the remote computer, your input from the keyboard goes to that system. Following is a list of commands you can execute while connected to a remote computer.

~.	Terminate the link.
~!	Escape to the local computer without dropping the link. To return to the remote computer, type Ctrl-D.
~!*command*	Execute *command* on the local computer.
~$*command*	Run *command* locally and send its output to the remote system.
~%**cd** *path*	Change directory on the local computer.
~%**take** *from* [*to*]	Copy a file from the remote computer to the local computer. If you don't specify a name for the copied file, it will have the same name as the original file. For example, the following command copies the *proposal* file from your current directory on the remote computer and places it in your current directory on the local computer. Because we did not specify a filename for the new file, it will also be called *proposal*. $ **~%take proposal** Return

~%put *from* [*to*]	Copy a file from the local computer to the remote computer. If you don't specify a name for the copied file, it will have the same name as the original file. For example, to copy a file named *minutes* from your current directory on the local computer to your current directory on the remote computer, you would type this command: $ **~%put minutes minutes.9-18** Return On the remote computer, the copy of the *minutes* file will be named *minutes.9-18*.
~~...	Send a line beginning with a tilde (~) to the remote computer.
~%break	Transmit a BREAK to the remote computer (can also specify as **~%b**).
~%nostop	Turn off the handshaking protocol for the remainder of the session. This is useful when the remote computer does not respond properly to the protocol characters.
~%debug	Toggles the **–d** debugging option on or off (can also specify as **~%d**).
~t	Display the values of the terminal I/O (input/output) structure variables for your terminal (useful for debugging).
~l	Display the values of the terminal I/O structure variables for the remote communication line (useful for debugging).

NOTE

The use of ~%**put** *requires that the* **stty** *and* **cat** *commands be present on the remote computer. Also, the erase and kill characters on the remote computer must be identical to the erase and kill characters on the local computer.*

The use of ~%**take** *requires the existence of the* **echo** *and* **cat** *commands on the remote computer. Also, if tabs are to be copied without expansion,* **stty tabs** *mode must be set on the remote computer.*

Other Information

cu(1C), **ls**(1), **sum**(1)

Appendix A
ASSIST Error Messages

Following are descriptions of the error messages that can occur when you run the ASSIST program.

ASSIST CANNOT FIND TERMINFO INFORMATION FILE FOR <*name*> TERMINAL TYPE

Your system's *terminfo* file does not contain information about the type of terminal you are using. Use **assist –s** to identify known *terminfo* entries.

ASSIST USAGE:

```
assist      Enter ASSIST at top-level menu
assist [command_name]      Enter command-form for command_name
assist -c [command_name]   Use current directory for data file
assist -s   Invoke ASSIST's setup activities
```

The **assist** command line did not have the proper syntax. Use one of the command lines listed above.

AT LEAST ONE OF THE HIGHLIGHTED ITEMS IS REQUIRED

Input is required in one of the highlighted fields.

AT LEAST ONE OF THE HIGHLIGHTED ITEMS ON THIS AND ON THE NEXT PAGE IS REQUIRED

Input is required in one of the fields highlighted on either this page or the next page.

BACKSPACE IS USED ONLY IN COMMAND FORMS (WHILE ENTERING AN INPUT STRING)

You are using Backspace on a menu. You can use Backspace only on a command form.

BACKSPACE IS USED ONLY WHEN ENTERING INPUT STRING

You can use Backspace on command forms only when you are entering input in a field. You cannot use Backspace to control cursor movement.

---

^C IS USED ONLY IN COMMAND FORMS (TO QUOTE CHARACTERS THAT ARE ASSIST COMMANDS)

You are using Ctrl-C on a menu. You can use Ctrl-C only on command forms.

CANNOT MOVE FARTHER TO LEFT

The cursor is positioned as far to the left as possible. You are trying to backspace beyond the possible limits.

CANNOT MOVE FARTHER TO RIGHT

The cursor is positioned as far to the right as possible. You are trying to space beyond the possible limits.

***x* CANNOT OCCUR BOTH IN CURRENT AND PREVIOUS ITEM**

The value entered in this field cannot be identical to the value in a previous field. Type a value that does not match the values in previous fields.

CONTAINS A *x*, MUST BE ENCLOSED BY *y*

The special character you entered must be enclosed in quotes.

DIRECTORY DOES NOT EXIST OR IS NOT ACCESSIBLE, TYPE <CR>

You are trying to move to a directory that does not exist or is not accessible because of access permissions. Change access permissions (use the **chmod** command) for the directory you want to move to or enter a different directory name.

DIRECTORY *x* HAS NO WRITE PERMISSION

You do not have write permission for the directory. Change directory permissions with the **chmod** command or enter a different directory name.

***x* DOES NOT EXIST**

The file or directory name printed in the message does not exist.

***x* DOES NOT HAVE EXECUTE OR SEARCH PERMISSION**

You do not have execute permission for the file or directory you named. You cannot execute this file or directory unless the permissions are changed to include execute permission.

DOES NOT HAVE READ PERMISSION

You do not have read permission on the file or directory you specified. You cannot read this file or directory unless the permissions are changed to include read permission.

***x* DOES NOT HAVE WRITE PERMISSION**

You do not have write permission on the file or directory you named. You cannot write in this file or directory unless the permissions are changed to include write permission.

***x* EXISTS**

You entered the name of an existing file or directory. If you continue with your command, the file will be overwritten. You can overwrite the file or type a different filename.

FILE *x* HAS NO WRITE PERMISSION

You do not have write permission for the file you specified. You cannot write in this file unless the permissions are changed to include write permission.

FIRST HAVE WORD VALIDATED BY STRIKING <CR>

Before backing up the cursor, you must press Return to validate the last piece of data entered. After the validation completes, you can back up the cursor.

***x* HAS EXECUTE OR SEARCH PERMISSION**

This message verifies that you have execute permission for the file or directory you specified.

***x* HAS MORE THAN *n* CHARACTERS**

The string you entered has too many characters.

***x* HAS READ PERMISSION**

This message verifies that you have read permission for the file or directory you specified.

***x* HAS WRITE PERMISSION**

This message verifies that you have write permission for the file or directory you specified.

INCOMPATIBLE WITH HIGHLIGHTED ITEM(S)

The function of the option you selected is incompatible with the highlighted option(s).

INCOMPATIBLE WITH ITEM(S) ON OTHER PAGE(S)

The function of the option you selected is incompatible with the options on another page.

INPUT *x* AND TARGET *y* ARE THE SAME

You have specified the same file or directory as both the input and output. The input file or directory cannot also be the output file or directory.

INPUT REQUIRED HERE

This field requires input. You must enter a value in this field.

INPUT SHOULD BE *a, b, c,* **OR** *z*

The value you entered is not valid. The error message lists all acceptable values.

INPUT SHOULD BEGIN WITH *string1, string2, ...* **OR** *stringn*

The input must begin with one of the strings listed in the error message.

INPUT SHOULD NOT BE *a, b, c,* **OR** *z*

The input for this field cannot include the values listed in the error message.

INPUT SHOULD NOT BEGIN WITH *string1, string2, ...* **OR** *stringn*

The input for this field cannot begin with the strings listed in the error message.

INPUT SHOULD END WITH *string1, string2,* **1... OR** *stringn*

The input for this field must end with one of the strings listed in the message.

INPUT SHOULD NOT END WITH *string1, string2, ...* **OR** *stringn*

The input for this field cannot end with one of strings listed in the error message.

INPUT STRING MUCH TOO LONG

Your entry exceeds the maximum number of characters allowed for this field.

---

x IS A DIRECTORY

You have specified a directory. A directory name is not the expected input.

---

x IS NOT A DIRECTORY

The name you entered is not a directory.

---

x IS NOT A VALID ASSIST COMMAND

You entered an invalid character in a first-letter search. The letters a-z and A-Z are the only valid characters.

---

^K IS USED ONLY IN COMMAND FORMS (TO STORE COMMAND LINE IN A FILE)

You used Ctrl-K on a menu. Ctrl-K is valid only on command forms.

NO ITEM ON THIS MENU STARTS WITH x

The letter you typed does not correspond to a menu item.

NOT A VALID COMMAND ON POPUP MENU

The command you entered is not valid on the pop-up menu.

NOT ACCEPTABLE; CHECK ITEM HELP

You have entered an invalid string. The list of acceptable strings or formats is too long to list in the error message. Refer to the item help for more information (press Ctrl-Y or F6).

NOT USED IN MENUS

The command you typed is not used on menus.

NOTHING TO ERASE

On a command form, Spacebar erases the string at the cursor position. You pressed Spacebar when the cursor was on a null string. Therefore, there is nothing to erase.

ONE STRING ONLY

ASSIST expects only one string in this field. You tried to enter more than one string. If your string contains a space, surround the string with quotes.

REQUIRED BY HIGHLIGHTED ITEM(S)

The highlighted items on the command form require that you enter data in this field.

REQUIRED BY HIGHLIGHTED ITEM(S) ON OTHER PAGE

The highlighted item(s) on another page of the command form require that you enter data in this field.

REQUIRES HIGHLIGHTED ITEM(S)

This field requires input entries in other fields. The fields where you must make an entry are highlighted.

REQUIRES ITEM(S) ON OTHER PAGES

This field requires input entries on other pages of the command form. You must enter input in these fields.

x* SHOULD BE *y

You typed *x*. You need to type *y* instead.

SHOULD BE *x*-SEPARATED LIST OF: *y,z,...*

Either the items on your list are separated by the wrong character or one or more members of the list are invalid.

<SPACE> IS USED ONLY IN COMMAND FORMS

You pressed Spacebar when you were not on a command form. Spacebar is valid only on command forms.

THERE IS ONLY ONE PAGE

You tried to go to the next page, however there is only one page.

TYPE NAME OF EXISTING DIRECTORY

ASSIST expects the name of an existing directory. You entered a non-existing name.

VALIDATION INTERRUPTED

There was an interruption while ASSIST was validating your input on a command form. You might have accidentally pressed Delete or Break.

WOULD OVERWRITE *x*

You have typed the name of an existing file or directory. If you continue your command, the contents of this file or directory will be destroyed.

YOU DO NOT OWN *x*

You are trying to perform an operation on a file or directory that you do not own. Only the owner of the file or directory can run that command.

Glossary

Alias. A word that represents login names or commands. In the **mailx** program, an alias represents one or more user logins. Mail sent to the alias will go to all users defined by the alias. In the Korn and C shell programs, an alias represents a command or program. For example, you could create an alias to run a command with a specific option.

Argument. An element of a command line that specifies the data on which a command is to operate (usually the name of a file or directory).

Background. A mode for executing commands. When you run a command in the background, the shell does not wait until the command completes. Instead, it displays a command prompt, allowing you to enter another command.

Bourne shell. The default shell program (**/bin/sh**).

Buffer. A temporary storage area in the computer's memory. The **ed** and **vi** editors store a copy of the file you are editing in a buffer.

C shell. A shell program (**/bin/csh**) that provides features such as job control and command history.

Child directory. See *subdirectory*.

Command. The name of an operating system file containing a program. When you issue a command, you type the name of a file. The operating system then executes the program located in that file.

Command interpreter. A program that reads your commands and changes them into language the computer understands. The shell is a command interpreter.

Command line. A command typed at the shell prompt (`$` or `%`). It can contain options and arguments and must end with Return.

Command mode. A mode used by the **ed** and **vi** editors and the **mailx** program. When **ed** or **vi** is in command mode, you can enter commands to move the cursor or edit text. When **mailx** is in command mode, you can read and respond to your mail.

Command substitution. A shell feature that allows you to use the output from one command as an argument to another command.

Control character. A character typed by holding down Ctrl while you press another key. These characters, which do not appear on the screen, are used for special purposes such as ending command input.

.cshrc. A start-up file used by the C shell. It contains aliases and other commands that affect your environment; it is executed whenever a new copy of the shell program starts up.

Current directory. The directory in which you are presently working. In pathnames, a dot (.) represents the current directory.

Current line. In editing programs, the current line is the line where the cursor is located.

Cursor. An indicator, usually a rectangle or blinking underscore, that shows your location on the screen.

Default. A value assigned by the operating system.

Delimiter. A character that separates words or arguments on a command line or in a file. Spaces and tabs are examples of delimiters.

Directory. A type of file that groups and organizes other files. You can create directories, add files to them, delete existing files, or remove the directories themselves.

Disable. Make unavailable. Printers can be disabled, and variables can be disabled.

Editor. A program that allows you to create and edit text.

Electronic mail. A program that allows you to send messages to other users. The operating system includes two mail programs: **mail** and **mailx.**

Enable. Make available. Printers and variables can be enabled.

Environment. The conditions in effect when you are working with the operating system. When you log in, the shell sets up an environment for you. This environment defines variables such as the type of terminal you are using, the search path used to locate commands, the command prompt that appears on the screen, and the name of your home directory. You can use the default environment, or you can modify it to meet your individual needs.

Environment variable. A variable that defines your environment. Some examples are **PATH**, which defines the search path used to locate commands, and **HOME**, which contains the name of your home directory.

Erase key. The key that deletes the previous character you typed. The default key is #; it is commonly changed to BackSpace.

Execute. The act of running a program or command.

Execute permission. For a directory, execute permission allows you to use the **cd** command to move to that directory. For a file, execute permission allows you to execute the program contained in that file.

Export. Make the value of a variable available to other programs.

Field. A specific part of a line. Commands such as **sort** and **cut** can look at individual fields on a line.

Field separator. A character such as a tab or space that separates the fields on a line.

File. A collection of information in the form of a stream of characters. Files can contain data, programs, or other text.

File access. The permissions that determine who can read, write, or execute your files.

Filename expansion. A shell feature that allows you to use metacharacters on the command line to match filenames. Also called *filename substitution*.

Filename extension. The characters in a filename that follow a period. Also called a *suffix*.

Filesystem. A set of files organized in a hierarchical structure. The root directory is the starting point for the filesystem. All other files and directories are arranged under the root directory.

Folders. A series of files used to save the mail you receive.

Foreground. A mode of command execution that causes the shell to wait until a command executes before prompting you for another command.

Fork. The action of a process creating another process.

Full pathname. A pathname starting at the root directory and leading to a specific file or directory. For example, */user1/terri/letters/rose* and */usr/bin* are full pathnames.

Global. A term that indicates a complete file. When a command is executed globally, it affects the entire file.

Group. A set of users defined by the system administrator. Each user belongs to a group. You can set file access permissions that apply only to members of your group.

History. A list of the commands you enter. The Korn and C shells allow you to reexecute commands from your history list.

Home directory. The directory where you are placed when you log in. It is usually named for your login name and is also called your login directory.

Input. The data a command acts on. The data can come from the terminal, a file, or another command.

Input mode. An operating mode used by the **ed** and **vi** editors and **mailx**. The **ed** and **vi** input modes allow you to add text to a file. The **mailx** input mode allows you to create and send messages.

Interactive. Allowing communication between you and the computer. The shell is interactive.

Job control. A feature of the Korn and C shells that allows you to stop or kill commands and to restart stopped commands either in the foreground or the background.

Kernel. The part of the operating system that controls access to the computer, handles hardware resources, manages the computer's memory, maintains the filesystem, and allocates computer resources among users.

Kill. To terminate a command.

Kill key. The key that deletes the command line you are typing. The default is @.

Korn shell. A shell program (**/bin/ksh**) that provides features such as job control, command line editing, and aliases.

Line address. The location of a line in an editing buffer (used by **ed** and **vi**).

Line number. The **ed** and **vi** editors assign numbers to each line in the buffer. You can display these numbers on the screen.

Link. A filename that refers to another file. You can use the **ln** command to create multiple filenames (typically in different directories) for a single file.

Logging in. Starting up a work session with the computer. Before you can use the operating system, you must type your login name and password. The system uses this information to verify that you are an authorized user.

.login. A C shell start-up file that executes when you log into the system. It contains commands and variables that affect your environment.

Login account. Information the operating system stores about you. The information includes your login name, home directory, password, group, and your default shell program. It can also include some miscellaneous information such as your phone number.

Login name. A name that identifies you to the computer. You must enter this name when you log into the system.

Logout. Exiting the operating system.

Mailbox. A file containing the mail messages sent to you.

.mailrc. A start-up file in your home directory that customizes the **mailx** program to meet your individual needs.

Metacharacters. Special characters used for matching filenames on the command line.

Modem. A device that connects a terminal and computer via telephone lines.

Octal notation. A system of notation that uses base 8. The permissions for a file or directory can be displayed as three octal digits. When you use the **chmod** command to change permissions, you can use octal notation to specify the appropriate permissions for the owner, group, and all other users.

Operating system. The software that controls the computer and your interaction with it.

Option. An element of a command line that modifies how the command runs. Most options are preceded by a hyphen (–), such as **–l**.

Ordinary file. A collection of characters the operating system treats as a unit. An ordinary file can contain text, code for a program, or any other information you want to save.

Output. The response the system generates when you issue a command. Output can print on your screen or you can redirect it to a file or another program.

Owner. The person who created a file. The owner of a file controls access to the file.

Parent directory. The directory immediately above a directory in the filesystem structure. In pathnames, this directory is represented by two dots (..).

Password. A code word that identifies you to the computer. You must enter this word when you log into the operating system.

Pathname. A name identifying the location of a specific file or directory in the filesystem. Pathnames beginning with the root directory are called *full pathnames*; pathnames beginning with your current directory are called *relative pathnames*.

Permissions. A set of numbers or characters that determine who can read, write, and execute your files. The owner of a file or directory can assign these permissions for the owner, the owner's group, and all other users.

PID. See *Process identification number.*

Pipe. A symbol (|) telling the operating system to use the output from one command as the input for another command.

Pipeline. A command line that contains a pipe.

Print request. A command to print the contents of a file on a printer.

Printer class. A group of printers. When you send a print request to a class, the first available printer in the class will print it.

Process. The execution of a command.

Process identification number (PID). A number the operating system assigns to each process. You can use PID numbers to display the status of processes or to kill them.

.profile. A Bourne and Korn shell start-up file that modifies your environment. You can add environment variables and commands to this file.

Program. A file that can be executed.

Prompt. The character the shell and other programs such as **mailx** use to indicate that you can enter a command.

Public directory. A directory that can be accessed by all users on a system. Typically, the */tmp* and */usr/tmp* public directories are used for temporary file storage. The UUCP system provides a public directory, */usr/spool/uucppublic*, where users from remote systems can deposit files.

Read permission. For a directory, read permission allows you to run the **ls** command to list the contents of the directory. For a file, read permission allows you to issue a command such as **cat** to read the contents of the file.

Redirection. The process of changing either the source for command input or the destination for command output.

Regular expression. A pattern used by programs such as **ed**, **vi**, **grep**, and **pg** to locate specific character strings in a file. A regular expression can contain special characters that match text in the file.

Relative pathname. A pathname starting in your current directory and leading up or down through a series of directories to a particular file or directory. For example, *../terri/memos* and *peterson* are relative pathnames.

Root directory. The source of the filesystem. This directory is represented by a slash (/).

Search path. A list of directories the shell searches to locate a command.

Scroll. The process of moving lines of text up or down on the screen.

Sequential execution. The execution of multiple commands that are typed on the same line. The commands must be separated by semicolons and will execute in the order they appear on the line.

Shell. A program that allows you to communicate with the operating system. It interprets your commands and is responsible for their execution. It also controls your environment and contains a programming language.

Special character. A character that tells the shell to perform a task such as redirecting input or output or matching filenames on the command line.

Special file. A file that represents a physical device such as a printer, terminal, or disk drive. When the system reads and writes to these files, the read and write requests activate the device handler associated with the file.

Spool. The practice of queuing a job for completion in the background. Print requests and UUCP jobs are spooled.

String. A series of characters.

Subdirectory. A directory located beneath another directory in the filesystem. Also called a *child directory*.

Superuser. A privileged user who owns most system files and uses the login **root**. The superuser can access any file, regardless of its permission levels, and performs most system administration tasks.

Symbolic address. A character representing a specific line in a text file. Used by the **ed** and **vi** editors.

Syntax. The format for typing a command line. The *Reference Manual* describes the syntax for operating system commands.

System administrator. A person responsible for the overall operation of the system, including adding users and devices, performing configuration tasks, and monitoring system activity.

System directory. A directory containing files used by the operating system. Some examples are */dev*, */etc*, */usr*, and */bin*. */dev* contains special files for devices, */etc* contains system administration files, and */usr* and */bin* contain programs.

UUCP. A collection of utilities that allow computers to communicate with each other over telephone lines, local area networks, or direct links. UUCP is used mainly to exchange mail.

Variable. A name that is assigned a value, either on the command line or in a file.

Wildcard. See *metacharacters*.

Working directory. Your current directory.

Write permission. For a directory, write permission allows you to create new files in the directory and to remove existing files. For a file, write permission allows you to edit the file.

Index

| - use command output as input 4-11
& - run command in background 4-6
& - match characters 8-32
? - match characters 4-3
* - match characters 3-44; 4-4; 8-30
! - match characters 4-6
; - execute multiple commands 4-7
^ - match characters 3-44; 8-31,34
< - redirect input 4-8
> - redirect output 4-9
>> - append output 4-10
[] - match characters 3-45; 4-5; 8-33
$ - variable substitution 4-18
$ - match characters 3-45; 8-32
. - match characters 3-44; 8-30

A

Absolute pathname 3-3
alias command 4-33,46
Aliases
  C shell 4-46
  Korn shell 4-33
  mailx 10-40
Ampersand 4-6; 8-32
Append output to a file 4-10
apropos command 5-1
Argument 1-6
ASSIST ..... 5-6 to 27
  change directory 5-26
  command forms ..... 5-14 to 23,25
  command search 5-25
  commands 5-9
  edit fields 5-21
  enter input 5-20
  error messages ..... A-1 to 8
  execute commands 5-22
  exit 5-26

function keys 5-7
invoke 5-7
menus ..... 5-8 to 9,11 to 12
pop-up help 5-12,18
pop-up menu 5-24
return to shell 5-26
save command line 5-22
set up 5-7
validate data 5-21
walkthrus 5-23
Asterisk
filename substitution 4-4
regular expressions 3-44; 8-30
at command 7-8

B

Background 4-6; 7-6
process status 7-2
Backquotes 4-14,17
Backslash 4-15,19
BackSpace 2-3
batch command 7-7
bg command 7-6
/bin directory 1-4
Bourne shell 4-1
display process status 7-2
kill processes 7-3
Brackets
filename substitution 4-5
regular expressions 3-45; 8-33
Buffer 8-1,9; 9-2

C

C shell 4-1
aliases 4-46
display process status 7-4
job control 7-4
kill processes 7-5
manage processes ..... 7-4 to 6
restart processes 7-6

stop processes 7-5
tilde substitution 4-49
cancel command 6-14
Cancel mail 10-12
Cancel print requests 6-14
Caret, regular expressions 3-44; 8-31,34
cat command 3-21
cd command 3-13; 4-21,34,38
cdpath variable 4-39
CDPATH variable 4-20
Change
directories 3-13
ASSIST 5-26
file and directory permissions ..... 3-35 to 37
groups 3-58
passwords 2-8
text, **ed** 8-19
Child directory 3-1
chmod command ..... 3-35 to 37
Clear window 9-45
Command forms
ASSIST ..... 5-14 to 23,25
Command history
C shell 4-47
display 4-31,47
edit a command 4-32
Korn shell 4-31
reexecute a command 4-32,47
search for a command 4-33,48
Command line 1-6
edit 4-30,32
Command mode
ed 8-1
mailx ..... 10-20 to 32
vi 9-2
Command substitution 4-14,36
Commands ..... 1-5 to 8; 2-10
add to *.login* 4-45
add to *.profile* 4-26
ASSIST 5-9
customize 4-33,46
execute on remote system 10-53

incorporate output in mail 10-16
print output 6-9
redirect input 4-8
redirect output 4-9
redirect output to another command 4-11
reexecute 4-32,47
run after logout 7-10
run at a later time ..... 7-7 to 10
run in background 4-6
Search path 4-24,42
type multiple commands 4-7
use output as argument 4-14
Communicate between sites 10-42
Connect remote system 10-56
Connect remote terminal 10-54
Control characters 2-4
Copy files 3-28
Copy text
ed 8-36
vi 9-42
Correct typing errors 2-2; 8-20; 9-16
Count lines, words, characters 3-38
cp command 3-28
Create
directories 3-12
files 3-21
links 3-31
processes 7-1
text
ed 8-18
vi 9-14
.cshrc file 4-42
ct command ..... 10-54 to 55
cu command ..... 10-56 to 60
Current directory 3-3,14
Cursor movement, **vi** ..... 9-17 to 25
Customize commands 4-33,46
Cut and paste text 9-41
cut command 3-56
cwd variable 4-39

D

date command 2-10
dead.letter file 10-5,12
Debug pipelines 4-12
Default destination 6-1,10
Delete
  directories 3-15
  files 3-33
  mail 10-7,26
  text
    ed 8-21
    vi ..... 9-30 to 34
/dev directory 1-5
diff command 3-40
Differences between files 3-40
Directories 1-3
  /bin 1-4
  /dev 1-5
  /etc 1-5
  /lib 1-5
  /tmp 1-5
  /usr 1-5
  access 3-7
  change 3-13
  change permissions 3-35
  child 3-1
  create 3-12
  current 3-3,14
  delete 3-33
  home 1-3; 3-1
  list contents 3-16
  name 3-6
  parent 3-1,4
  permissions 3-7
  remove 3-15,33
  root 1-4; 3-1
  system 3-1
Disk quotas ..... 3-59 to 62
Display
  command history 4-31,47
  file contents 3-21,23,27

filenames, **ed** 8-38
mailx variables 10-32
printing status ..... 6-10 to 13
process status
Bourne shell 7-2
C shell 7-4
Korn shell 7-4
text 8-12
variables 4-22,40
Dollar sign
regular expressions 3-45; 8-32
variable substitution 4-18
Dot command 4-25
Double quotation marks ..... 4-17 to 19

E

echo command 4-13,23,41
ed
change text 8-19
command prompt 8-9
commands 8-2
copy lines 8-36
correct typing errors 8-20
create text 8-18
delete lines 8-21
display filenames 8-38
display text 8-12
edit **mailx** messages 10-13
help messages 8-17
invoke 8-8
join lines 8-36
markers 8-40
move lines 8-35
quit 8-11
read in a file 8-37
return to shell 8-39
sample exercises ..... 8-2 to 7
save text 8-10
search for text ..... 8-22 to 25
special characters ..... 8-29 to 35
substitute text 8-25

system interrupts 8-39
undo commands 8-21
Edit command lines 4-30,32
Edit mail 10-13,30
Edit multiple files 9-10
EDITOR variable 4-21,38
Electronic mail ..... 10-4 to 40
Enter commands 2-10
Enter text, **vi** 9-15
ENV variable 4-29
Environment ..... 4-1,20 to 27,37 to 46; 9-2
Environment variables ..... 4-20 to 25,38 to 44
CDPATH 4-20
display value 4-23,41
EDITOR 4-21,38
ENV 4-29
FCEDIT 4-29,32
HISTFILE 4-29
HISTSIZE 4-29
HOME 4-21,38
IFS 4-21
incorporate in mail 10-15
Korn shell 4-29
LOGNAME 4-21,38
LPDEST 4-21,38
MAIL 4-21,39
MAILCHECK 4-21
MAILPATH 4-21
mailx ..... 10-32 to 40
modify 4-23,41
OLDPWD 4-29,35
PARALLEL 4-21,39
path 4-42
PATH 4-21,24,39
PS1 4-21,23,29
PS2 4-21
PWD 4-29,35
SHACCT 4-22
SHELL 4-2,22,39
TERM 4-22,25,39,43; 9-2
TERMINFO 4-22,39
TZ 4-22,39

USER 4-39
VISUAL 4-22,30,39
Erase key 2-3; 4-26,45; 8-20; 9-16
print as text 2-3
Error messages, ASSIST A-1
/etc directory 1-5
Execute *.login* 4-44
Execute *.profile* 4-25
Execute commands 1-7
ASSIST 5-22
Execute commands on a remote system 10-53
Execute permission 3-9,35
Exit ASSIST 5-26
exit command 2-11
Export variables 4-20,23,24
.exrc file ..... 9-2,48 to 51
Extract fields or characters 3-56

F

fc command 4-32
FCEDIT variable 4-29,32
fg command 7-6
Field delimiter 3-52
Field separator 3-56
Fields 3-52,56
File access
change 3-35
Filenames
expansion 4-2
extension 3-7
select 3-6
substitution ..... 4-2 to 6
suffix 3-7
Files
access 3-7
change permissions 3-35
copy 3-28
count lines, words, characters 3-38
create 3-21
.cshrc 4-42
cut fields or characters 3-56

dead.letter 10-5,12
delete 3-33
display contents 3-21,23,25,27
/etc/profile 4-20,38
.exrc ..... 9-2,48 to 51
.history 4-31,47
identify differences 3-40
links 3-31
locate 3-46
.login 4-42
mail to another user 10-9,31
.mailrc ..... 10-34 to 40
mail.rc 10-32
mailbox 10-5,10
mbox 10-10,25
merge 3-50
move 3-30
name 3-6
ordinary 1-3
paginate 3-25
permissions 3-7
print 6-2
.profile ..... 4-20,24 to 27,38; 10-3
remove 3-33
rename 3-30
search for a pattern 3-42
sort 3-49
special 1-3
type 3-8,19
Filesystem ..... 1-3 to 5; 3-1 to 10
find command ..... 3-46 to 48
Foreground 4-6; 7-6
Forward mail 10-8,14
Full pathname 3-3
Function keys, ASSIST 5-7

G

Global search 8-24
Global substitution 8-28; 9-40
glossary command 5-5
Greater-than sign 3-11; 4-9

grep command ..... 3-42 to 45
Groups 3-8
  change 3-58
groups command 3-58

H

Hard limit 3-59
help command 5-3
Help messages, **ed** 8-17
HISTFILE variable 4-29
history command 4-31,47
history variable 4-40
History, commands 4-31,47
HISTSIZE variable 4-29
Home directory 1-3; 3-1
home variable 4-40
HOME variable 4-21,35,38

I

IFS variable 4-21
ignoreeof variable 4-40
Input mode
  ed 8-1
  mailx ..... 10-12 to 19
  vi 9-2,14
Input, redirect 4-8; 10-9,31

J

Job control 7-4
jobs command 7-4
Join lines
  ed 8-36
  vi 9-45

K

Kernel 1-1
Keyboard characteristics 2-2

kill command
Bourne shell 7-3
C shell 7-5
Korn shell 7-5
Kill key 2-2; 4-26,45; 8-20; 9-16
print as text 2-3
Kill processes
Bourne shell 7-3
C shell 7-5
Korn shell 7-5
Korn shell ..... 4-1,28 to 36
aliases 4-33
cd command 4-34
command substitution 4-36
display process status 7-4
edit command line 4-30
invoke 4-28
job control 7-4
kill processes 7-5
manage processes ..... 7-4 to 6
restart processes 7-6
startup file 4-29
stop processes 7-5
tilde substitution 4-35

L

Less-than sign 4-8
Letter case, change in **vi** 9-45
/lib directory 1-5
Line address 8-2
Line numbers 8-14; 9-26
Links ..... 3-20,31 to 32
List directory contents 3-16
ln command ..... 3-31 to 32
locate command 5-4
Locate files 3-46
Log in ..... 2-4 to 7
Log off 2-11
.login file 4-42

Login
account 2-4; 4-28
name 2-5
password 2-6
process 2-5
LOGNAME variable 4-21,38
lp command ..... 6-2 to 10
LPDEST variable 4-21,38
LP print service 6-1
lpstat command ..... 6-10 to 13
ls command ..... 3-7,16 to 20

M

mail ..... 10-4 to 9
delete message 10-7
forward messages 10-8
mail files 10-9
quit 10-8
read mail 10-5
return to shell 10-8
save messages 10-7
send mail 10-4
send mail via UUCP 10-43
Mail files 10-9,31
MAIL variable 4-21,39
Mailbox 10-5
MAILCHECK variable 4-21
MAILPATH variable 4-21
mail.rc file 10-32
.mailrc file ..... 10-34 to 40
mailx ..... 10-10 to 40
aliases 10-40
cancel messages 10-12
change message header 10-16
change variables 10-33
copy messages 10-28
delete messages 10-26
display header information 10-24
display variables 10-32
edit messages 10-13,30
environment ..... 10-32 to 40

exit 10-23
files 10-10
forward messages 10-14
header information 10-20
hold messages in mailbox 10-28
incorporate an environment variable 10-15
incorporate command output 10-16
incorporate mail 10-14
mail files 10-25,31
message list 10-21
pipe a message through a command 10-30
preview messages 10-13
read in a file 10-14
read mail ..... 10-20 to 23,25
read *mbox* file 10-25
reply to mail ..... 10-29 to 30
return to shell 10-18,31
save messages 10-19,27
send mail via UUCP 10-43
signature 10-18
write messages 10-10
man command 5-1
Man pages, online 5-1
Markers 8-40; 9-47
Match filenames ..... 4-2 to 6
Match patterns
ed ..... 8-29 to 35
grep 3-43
pg 3-24
vi 9-29
mbox file 10-25
Menus, ASSIST ..... 5-8 to 9,11 to 12
Merge files 3-50
mesg command 10-3
Metacharacters 4-2
mkdir command 3-12
Modify environment variables 4-23,41
Move files 3-30
Move text 8-35
mv command 3-30

N

Name files and directories 3-6
Named registers, **vi** 9-43
newgroup command 3-58
news command 10-1
NFS and quotas 3-61
noclobber variable 4-40
nohup command 7-10
Non-printing characters 8-38

O

Octal notation 3-36
OLDPWD variable 4-29,35
Online man pages 5-1
Operating system 1-1
  walkthru 5-23,27
Option 1-6
Ordinary files 1-3
Output
  append to existing file 4-10
  incorporate in mail 10-16
  redirect to another command 4-11
  redirect to file 3-11; 4-9
  redirect to printer 6-9
  use as argument to another command 4-14

P

Paginate and display files 3-25
PARALLEL variable 4-21,39
Parent directory 3-1,4
passwd command 2-9
Passwords ..... 2-7 to 9
Path prefix 3-4
path variable 4-40,42
PATH variable 4-21,24,39
Pathnames ..... 3-3 to 6
  full 3-3
  relative 3-4
Percent sign, regular expressions 8-33

Period, regular expressions 3-44; 8-30
Permissions ..... 3-7 to 10,20
  change ..... 3-35 to 37
pg command ..... 3-23 to 24
PID 7-1
Pipeline 4-11
  debug 4-12
Pipes 4-11; 6-9; 10-30
Pop-up help, ASSIST 5-12,18
Pop-up menu, ASSIST 5-24
Post news items 10-2
pr command 3-25
Print arguments 4-13
Print requests 6-2,11
Printers 6-1
  class 6-1
  default destination 6-1,10
  disable 6-16
  display print requests 6-11
  enable 6-15
  status 6-12
Printing
  banner page 6-5
  cancel requests 6-14
  command output 6-9
  copy before printing 6-3
  display status 6-10
  files 6-2
  multiple copies 6-4
  notify when job complete 6-4
  specify printer 6-3
Processes 7-1
  Bourne shell
    display status 7-2
    kill 7-3
  C shell
    display status 7-4
    kill 7-5
    restart 7-6
    stop 7-5
  create 7-1
  identification number 7-1

Korn shell
display status 7-4
kill 7-5
restart 7-6
stop 7-5
.profile file ..... 4-24 to 27; 10-3
Program 7-1
prompt variable 4-40
ps command 7-2
PS1 variable 4-21,23,29
PS2 variable 4-21
pwd command 3-3,14
PWD variable 4-29,35

Q

Question mark, filename substitution 4-3
Quit
ed 8-11
vi 9-11
quota command 3-60
Quota limits 3-59
Quotation marks 4-16

R

Read in a file
ed 8-37
vi 9-45
Read mail ..... 10-5,20 to 23
Read news 10-1
Read permission 3-9,35
Receive files via UUCP ..... 10-50 to 52
Redirect input 4-8
Redirect output 3-11; 4-9
to a printer 6-9
to an existing file 4-10
to another command 4-11
Redirection ..... 4-8 to 13
Reexecute commands 4-32,47
Regular expressions ..... 3-43 to 45; 8-22 to 35; 9-29
Relative address 8-16

Relative pathname 3-4
Remote system 10-56
Remote terminal 10-54
Remove directories 3-15
Remove files 3-33
Remove meaning from special characters ..... 4-14 to 19
Rename files 3-30
Repeat commands, **vi** 9-44
Replace text, **vi** 9-34
Reply to mail ..... 10-29 to 30
Restart processes 7-6
Return to shell
  ASSIST 5-26
  ed 8-39
  mail 10-8
  mailx 10-18,31
  vi 9-46
rm command 3-33
rmdir command 3-15
Root directory 1-4; 3-1
Run commands in background 4-6

S

Save command line, ASSIST 5-22
Save mail 10-7,19,27
Scroll text, **vi** 9-24
Search for a command 4-33,48
Search for a pattern
  ed 8-22
  grep 3-42
  pg 3-24
  vi 9-28
Search path 4-24,42
Security, files 3-7
Semicolon 4-7
Send mail 10-4
Send mail via UUCP 10-43
Send message to a terminal 10-2
Sequential execution 4-7
set command 4-22,40; 10-32
Set environment variables 4-23,41

Set **mailx** variables 10-33
Set terminal options 4-26,45
Set up ASSIST 5-7
SHACCT variable 4-22
Shell 1-2
Shell environment ..... 4-20 to 27,29; 9-2
shell variable 4-40
SHELL variable 4-2,22,39
Single quotation marks 4-16
Soft limit 3-59
sort command ..... 3-49 to 55
Sort files ..... 3-49 to 55
source command 4-44
Special characters 4-1
  command substitution 4-14
  execute commands sequentially 4-7
  filename substitution ..... 4-2 to 6
  in regular expressions ..... 3-43 to 45; 8-30 to 35; 9-29
  input and output redirection ..... 4-8 to 12
  remove meaning ..... 3-45; 4-14 to 19
  run commands in background 4-6
Special file 1-3
spell command 9-47
Spooling 6-1
starter command 5-4
Stop processes 7-5
stty command 4-26,45
Substitute output as an argument 4-14
Substitute text ..... 8-25 to 28
  vi 9-36
Substitute value for variable name 4-18
Superuser 3-10
Symbolic address ..... 8-14 to 16
Symbols for changing permissions 3-35
Syntax 1-6
System directories 3-1
System interrupts
  ed 8-39
  vi 9-10
System mailbox 10-5

T

Tabs 4-26,45
tail command 3-27
tee command 4-12
Term type 4-25,43
term variable 4-40
TERM variable 4-22,25,39,43; 9-2
TERMINFO variable 4-22,39
Terminal options 4-26,45
Terminals ..... 2-1 to 4
Tilde substitution 4-35,49
/tmp directory 1-5
Transfer files to remote systems 10-43
Transpose letters, **vi** 9-42
Typing errors 2-2; 8-20; 9-16
Typing speed 2-2
TZ variable 4-22,39

U

umask command 3-35
uname command 10-41
Underscore variable 4-29
Undo commands
  ed 8-21
  vi 9-30
unset command 10-33
usage command 5-5
User information 2-11
USER variable 4-39
/usr directory 1-5
UUCP ..... 10-41 to 60
  determine if sites communicate 10-42
  display job status 10-48
  execute commands 10-53
  find a site name 10-41
  receive files ..... 10-50 to 52
  send mail 10-43
  transfer files to remote systems 10-43
uucp command ..... 10-43,44 to 46
uuname command 10-42

uustat command 10-48
uuto command ..... 10-43,47,50 to 52
uux command 10-53

V

Validate data, ASSIST 5-21
Variable substitution 4-18,23,41
Variables 4-18
  cdpath 4-39
  cwd 4-39
  display value 4-23,41
  environment ..... 4-20 to 25,38 to 44
  history 4-40
  home 4-40
  ignoreeof 4-40
  mailx 10-32
  noclobber 4-40
  path 4-40
  prompt 4-40
  shell 4-40
  term 4-40
vi
  search patterns ..... 9-28 to 29
  change letter case 9-45
  change text ..... 9-37 to 41
  clear window 9-45
  command mode 9-2,4
  commands 9-1
  copy text 9-42
  correct typing errors 9-16
  create text 9-14
  cursor movement ..... 9-17 to 25
  cut and paste text 9-41
  delete text ..... 9-30 to 34
  edit command line 4-30
  edit **mailx** messages 10-13
  edit multiple files 9-10
  enter text 9-15
  environment ..... 9-48 to 51
  input mode 9-2,3,14
  invoke 9-3,9

join lines 9-45
line numbers ..... 9-26 to 28
markers 9-47
named registers 9-43
quit ..... 9-5,11 to 14
read in a file 9-45
repeat commands 9-44
replace text 9-34
return to the shell 9-46
sample sessions 9-3
scroll text 9-24
spelling errors 9-47
substitute text 9-36
system interrupts 9-10
transpose letters 9-42
undo commands 9-30
view files 9-11
walkthru 5-23
write to a file ..... 9-5,7,11 to 14
View files, **vi** 9-11
VISUAL variable 4-22,30,39

W-X-Y-Z

Walkthru, ASSIST 5-23
wc command 3-38
whatis command 5-1
who command 2-11
Wild cards 4-2
write command 10-2
Write mail 10-10
Write permission 3-9,35
Write to a file ..... 8-10; 9-11 to 14
Yank text 9-42